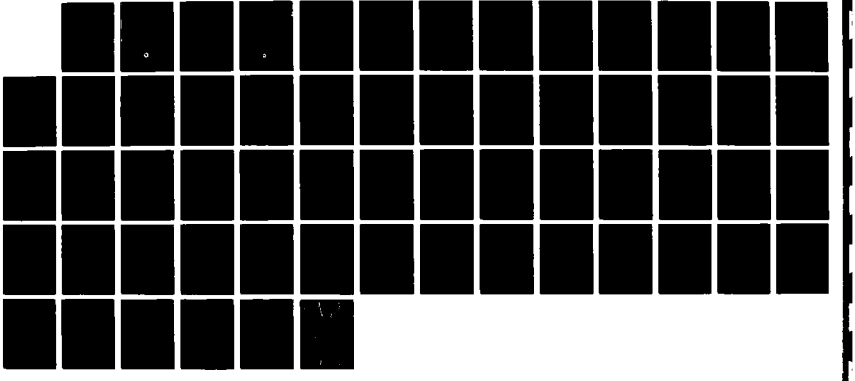
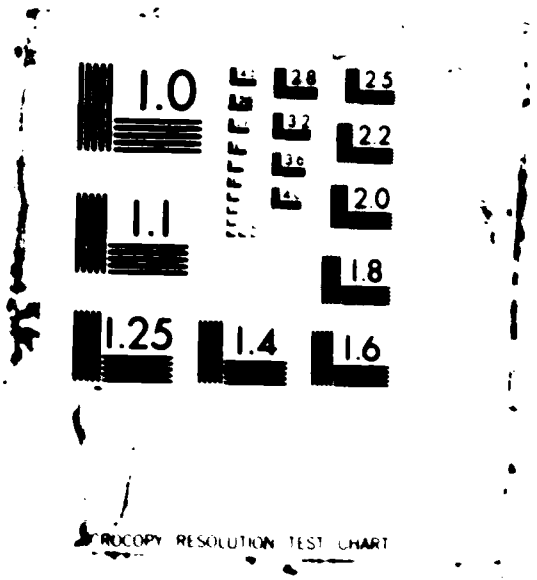
AD-A179 452 A VERSATILE TOOL FOR DATA FILE TRANSFER AND MANIPULATION(U) DEFENCE RESEARCH ESTABLISHMENT ATLANTIC DARTMOUTH (NOVA SCOTIA) J B FARRELL JAN 87 1/1

UNCLASSIFIED DREA-TC-87/303 F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
ROCOPY RESOLUTION TEST CHART

UNLIMITED DISTRIBUTION

National Defence
Research and Development Branch

Défense Nationale
Bureau de Recherche et Développment

(3)

TECHNICAL COMMUNICATION 87/303
January 1987

A VERSATILE TOOL FOR DATA FILE TRANSFER AND MANIPULATION

Joseph B. Farrell

Defence Research Establishment Atlantic

Centre de Recherches pour la Défense Atlantique

Canada

DEFENCE RESEARCH ESTABLISHMENT ATLANTIC

9 GROVE STREET
P O. BOX 1012
DARTMOUTH, N.S.
B2Y 3Z7

TELEPHONE
(902) 426.3100

CENTRE DE RECHERCHES POUR LA DÉFENSE ATLANTIQUE

9 GROVE STREET
C.P 1012
DARTMOUTH, N É
B2Y 3Z7

UNLIMITED DISTRIBUTION

National Defence
Research and Development Branch

Défense nationale
Bureau de recherche et développement

A VERSATILE TOOL FOR DATA FILE TRANSFER AND MANIPULATION

Joseph B. Farrell

January 1987

Approved by H.M. Merklinger H/Surveillance Acoustics Section

DISTRIBUTION APPROVED BY

D/UAD

TECHNICAL COMMUNICATION 87/303

Defence Research Establishment Atlantic

Centre de Recherches pour la Défense Atlantique

Canada

Abstract

This document describes in detail a software tool for manipulating data files. The Surveillance Acoustics section at Defence Research Establishment Atlantic has acquired VAX computers over the last few years, and analysis tasks which were formerly done on PDP-11 computers are now being moved to the VAXen. PDP-11s are still used in the at-sea data collection role, so some means is necessary of transferring the data files thus produced to the VAXen for signal processing and analysis. PDP-11 data files are typically located on 9-track magnetic tape, so one method of transferring the data would be to read PDP-11 tapes on the VAXen. The software tool described here (a program named **TRANSFER**) was written, in part, to perform this data transfer chore, taking into account the special formats and header information in the files produced by the PDP-11s. Manipulation of data files already residing on a VAX is also possible using TRANSFER. The program is versatile, allowing the user to choose channels and data segments to be transferred between files with a high degree of freedom.

Sommaire

Le présent article décrit en détail un outil logiciel permettant la manipulation de fichiers de données. La section de l'acoustique de surveillance du Centre de recherches pour la défense, Atlantique, s'est doté d'ordinateurs VAX au cours des dernières années et les travaux d'analyse effectués jusqu'à présent sur des PDP-11 le sont maintenant au moyen du VAXen. Toutefois, les PDP-11 sont encore utilisés pour la cueillette des données en mer; les travaux d'analyse et de traitment des données nécessitent donc une méthode de transfert sur le VAXen des fichiers produits pendant la cueillette des données. Puisque les bandes magnétiques de 9 pistes constituent le support de mémoire typique des fichiers de données PDP-11, on peut envisager la procédure suivante comme méthode de transfert des données: lecture des données mémorisées sur les bandes PDP-11, puis transfert au VAXen. L'outil logiciel décrit dans le présent texte (un programme appelé **TRANSFER**) a été écrit en partie en vue d'assurer cette tâche de transfert des données, en fonction des données d'entête et des formats spéciaux des fichiers produits par le PDP-11. TRANSFER permet aussi la manipulation des fichiers de données déjà mémorisés dans un ordinateur VAX. Ce logiciel est très polyvalent et permet à l'utilisateur de choisir avec un grand degré de liberté les pistes et les segments de données à transférer d'un fichier à l'autre.

Table Of Contents

1. INTRODUCTION

Until recently, data collected at sea by DREA scientists were analysed almost exclusively on PDP-11/34s. When the Surveillance Acoustics section began to purchase VAX computers, the situation changed drastically. Most analysis tools which existed on the 11/34s are now available on the VAXen (the accepted plural form for VAX) in a similar or more powerful form. New tools (such as a suite of shot-analysis programs) have also been appearing. The multi-user nature of the VAXen has made these software tools available to a wider range of users, and consequently more analysis is being performed.

PDP-11s have been used in the at-sea data collection role at DREA for approximately 10 years, and perform their job very efficiently. Real time data collection programs write data (generally to 9 track magnetic tape) in a format which has been well tuned to the needs of DREA scientists over the lifetime of the PDP-11s. Naturally enough, this format is somewhat foreign to the VAX computer, and data tapes produced on the PDP-11s cannot be read directly on the VAX without a software interface. To allow more users to access and analyze raw data, such a software interface has been written, and this note describes that tool, a program called **TRANSFER**.

TRANSFER has been written to be as general as possible, leaving many options open to the user. Most files written to tape by the PDP-11 data acquisition/analysis programs can be moved from mag-tape to the VAX using this program. The formats accepted are ".DAT" (time series data files), ".FTR" (Fourier coefficient files), and ".PWR" (power spectrum files). Other features such as transferring a segment of an input file defined by start and stop times, or transferring only a subset of the total number of channels in the input file are also available. Disk space is at a premium on the VAXen (as it is on all computers), so the latter feature is an important one. It allows disk space to be conserved if a user wishes to analyze only a few of the channels of input data available.

The default format for VAX disk files created by TRANSFER is binary direct access. This format allows random access to any block in the file, and is a relatively compact storage format. Most of the analysis programs now present on the VAXen accept input files in this format. An older format (READRT) is still used by some analysis programs, and TRANSFER will use this as its output format if requested.

Size reduction for disk files already on the VAX is desirable in many circumstances, so TRANSFER will also perform VAX disk file to VAX disk file transfers. The same options are offered in the disk-to-disk mode as in the tape-to-disk mode.

The next section of this note describes the formats of the tape files TRANSFER will accept from PDP-11s, and gives more details on the VAX and READRT disk file formats. Following that, a detailed description of program implementation will be given, including a discussion of all the options available. An example of program use is then presented to give the reader some feeling of how a terminal session proceeds and finally, some possibilities for future developments of the program are proposed.

2. FILE FORMATS

TRANSFER was originally written to accept 9-track magnetic tapes written by the DREA PDP-11 data recording programs, and transfer them to VAX files. To make this document self-contained, a brief description of the format of the various types of tape input

files will be presented here. A fuller description of the file formats and the philosophy behind their structure is contained in a Technical Communication by D. Caldwell [1]. Disk file types which are compatible with TRANSFER will also be described.

2.1 Time Series (.DAT) File Format

Time series data (.DAT) files are the most common type of input file used with the TRANSFER program. Time series data are recorded on magnetic tape on the PDP-11s in this format, and since it is the intention to use the VAX for most analysis, raw input data will be moved to the VAX via this file type.

Each tape file begins with a 512 byte header which describes the physical parameters of the file such as record length, sampling frequency, etc. The header is divided into four blocks - the first 32 bytes form an integer block (2 bytes per integer), the next 32 bytes form a floating point block (4 bytes per floating-point number), the next 128 bytes form a byte block (1 byte per entry) and the remaining 320 bytes form an ASCII block (1 character per byte). Table I illustrates these blocks and gives a brief description of the contents of each location in the header block. A more detailed description of the meaning of the header block contents can be found in [1].

The data portion of a typical time series file is diagrammed in Figure 1. In the sample file shown, there are m time samples for each of the n input channels. The values are multiplexed so that the first time sample for each channel appears in sequence, followed by the second time sample for each channel, etc. The DREA header actually allows the time series data to be written in other formats, but the one shown here is used almost exclusively.

2.2 Fourier Coefficient (.FTR) File Format

This type of file is used by analysis programs which require Fourier coefficients but do not contain an FFT module of their own. Programs for performing interference cancellation which require ".FTR" files currently exist on the VAX. A program (called SAFTR) [2] can be used to produce ".FTR" files from ".DAT" files on the VAX. TRANSFER will work with the ".FTR" format, but the ".DAT" format is more likely to be encountered.

As in the ".DAT" format, each ".FTR" file begins with a 512 byte header [1]. Table II gives a brief description of the contents of each location in the ".FTR" header block. Figure 2 shows the contents of a typical ".FTR" file. In the sample file shown, the data are multiplexed in a different manner from that in a ".DAT" file. Here, all Fourier coefficients from each transform of each channel are kept together; that is, blocks of data for each channel rather than single samples are multiplexed.

2.3 Power Spectrum (.PWR) File Format

A power spectrum tape file is likely to be transferred from tape only if time series analysis was performed on a PDP-11. SEQFFT is the most widely used spectral analysis program on the PDP-11s, and the output of that program conforms to the ".PWR" format. As analysis effort moves to the VAX from the 11s, transfer of this file type is likely to become less common.

The ".PWR" file begins with a 512 byte header [1]. TABLE III gives a brief description of the contents of each location in the ".PWR" header block. Figure 3 shows

INTEGER BLOCK

ILABEL(1)	= Block size	# of 16 bit words/physical block
ILABEL(2)	= Record Size	# of words per logical record
ILABEL(3)	= # of records	# of data records in file
ILABEL(4)	= Repetition rate	# of records per repetition cycle
ILABEL(5)	= Number type	Int=1,Flt=2,CmplxI=11oct,CF=12oct
ILABEL(6)	= Bytes per number	eg.: I=2,F=4,CI=4,CF=8
ILABEL(7)	= # of channels	Must divide evenly into ILABEL(2)
ILABEL(8)	= Multiplex length	Word=1, Record=Record size
ILABEL(9)	= # accumulations	Normally 1
ILABEL(10)	= X-axis	Time=1, Frequency=2
ILABEL(11)	= Y-axis	Linear=1,Square=2,Log=4
ILABEL(12)	= Sequence #	User assigned, usually increments
ILABEL(13)	= Block Scaling	Power of 2 scaling factor
ILABEL(14)	= Spare	
ILABEL(15)	= Spare	
ILABEL(16)	= Spare	

FLOATING-POINT BLOCK

FLABEL(1)	= Sampling freq(Hz)	-ve means heterodyned
FLABEL(2)	= Heterodyning freq.	-ve means real heterodyned
FLABEL(3)	= Reference level	Calibration factor
FLABEL(4)	= Max. magnitude	
FLABEL(5)	= Gain correction	1.00343332 for power of 2
FLABEL(6)	= Spare	
FLABEL(7)	= Spare	
FLABEL(8)	= Spare	

BYTE BLOCK

BLABEL(1)	= Channel #	
BLABEL(2)	= Gain (dB)	for channel in BLABEL(1)
BLABEL(3)	= Channel #	
BLABEL(4)	= Gain (dB)	for channel in BLABEL(3)
.		
.		
.		
BLABEL(127)	= Channel #	
BLABEL(128)	= Gain (dB)	for channel in BLABEL(127)

ASCII BLOCK

ALABEL(1)	= First character of ASCII label block	
.		
.		
.		
ALABEL(320)	= Last character of ASCII label block	

Table I: Header contents of a Time Series (.DAT) file

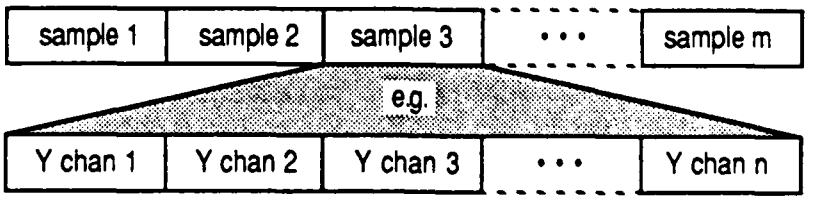


Figure 1: Data format in a typical Time Series (.DAT) file

INTEGER BLOCK

```
ILABEL(1)  = Block size          # of 16 bit words/physical block
ILABEL(2)  = Record Size         # of words per logical record
ILABEL(3)  = # of records        # of data records in file
ILABEL(4)  = Number of sequential transforms
ILABEL(5)  = Number type         Int=1,Flt=2,CmplxI=11oct,CF=12oct
ILABEL(6)  = Bytes per number    eg.: I=2,F=4,CI=4,CF=8
ILABEL(7)  = # of channels       Must divide evenly into ILABEL(2)
ILABEL(8)  = # of frequency bins
ILABEL(9)  = # accumulations     Normally 1
ILABEL(10)= X-axis               Time=1, Frequency=2
ILABEL(11)= Y-axis               Linear=1,Square=2,Log=4
ILABEL(12)= Sequence #           User assigned, usually increments
ILABEL(13)= Block scaling        Power of 2 scaling factor
ILABEL(14)= Window type          None=1,Hanning=2,Hamming=3,Kaiser=4
ILABEL(15)= # of zeros
ILABEL(16)= # of points of overlap
```

FLOATING-POINT BLOCK

```
FLABEL(1) = Start frequency of first bin (Hz)
FLABEL(2) = Heterodyning freq. -ve means real heterodyned
FLABEL(3) = Frequency resolution (Hz)
FLABEL(4) = Max. magnitude
FLABEL(5) = Spare
FLABEL(6) = % overlap
FLABEL(7) = Time interval of a simgle FFT (hours)
FLABEL(8) = Spare
```

BYTE BLOCK

```
BLABEL(1) = Channel #
BLABEL(2) = Gain (dB)
BLABEL(3) = Channel #
BLABEL(4) = Gain (dB)
    .
    .
    .
BLABEL(127)= Channel #
BLABEL(128)= Gain (dB)
```

ASCII BLOCK

```
ALABEL(1)   = First character of ASCII label block
    .
    .
    .
ALABEL(320)= Last character of ASCII label block
```

Table II: Header contents of a Fourier Coefficient (.FTR) file

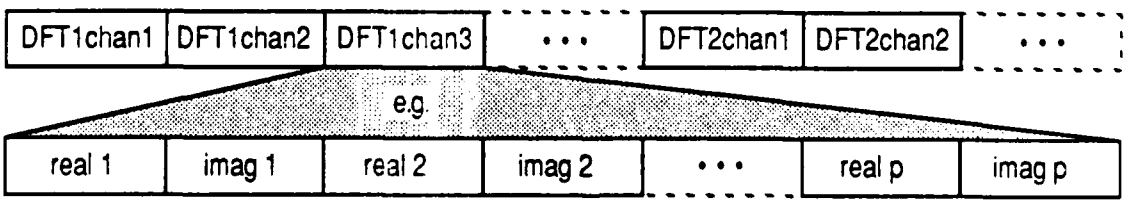


Figure 2: Data format in a typical Fourier Coefficient (.FTR) file

INTEGER BLOCK

```
ILABEL(1) - Block size          # of 16 bit words/physical block
ILABEL(2) - Record Size         # of words per logical record
ILABEL(3) - # of records        # of data records in file
ILABEL(4) - Number of sequential spectral estimates
ILABEL(5) - Number type         Int=1,Flt=2,CmplxI=11oct,CF=12oct
ILABEL(6) - Bytes per number    eg.: I=2,F=4,CI=4,CF=8
ILABEL(7) - # of channels
ILABEL(8) - # of frequency bins
ILABEL(9) - # accumulations     Normally 1
ILABEL(10)- X-axis              Time=1, Frequency=2
ILABEL(11)- Y-axis              Linear=1,Square=2,Log=4
ILABEL(12)- Sequence #          User assigned, usually increments
ILABEL(13)- Spare
ILABEL(14)- Window type         None=1,Hanning=2,Hamming=3,Kaiser=4
ILABEL(15)- # of zeros
ILABEL(16)- # of points of overlap
```

FLOATING-POINT BLOCK

```
FLABEL(1) - Center frequency of first bin (Hz)
FLABEL(2) - Heterodyning freq. -ve means real heterodyned
FLABEL(3) - Frequency resolution (Hz)
FLABEL(4) - Max. magnitude
FLABEL(5) - Spare
FLABEL(6) - Time interval between sequential frames (hrs.)
FLABEL(7) - % overlap
FLABEL(8) - Spare
```

BYTE BLOCK

```
BLABEL(1) - Channel #
BLABEL(2) - Normally 0.         gain already compensated for
BLABEL(3) - Channel #
BLABEL(4) - Normally 0
     .
     .
     .
BLABEL(127)- Channel #
BLABEL(128)- Normally 0
```

ASCII BLOCK

```
ALABEL(1)   - First character of ASCII label block
     .
     .
     .
ALABEL(320)- Last character of ASCII label block
```

Table III: Header contents of a Power Spectrum (.PWR) file

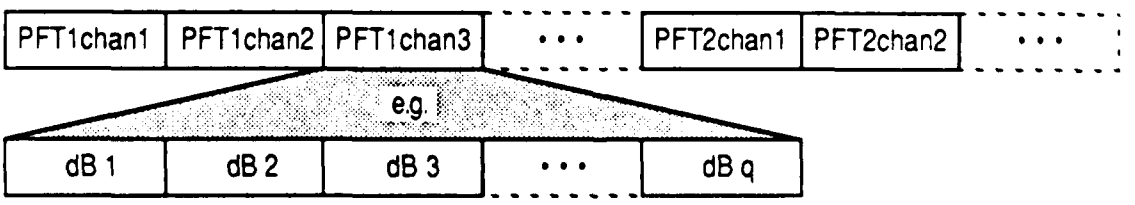


Figure 3: Data format in a typical Power Spectrum (.PWR) file

the contents of a typical ".PWR" file. Multiplexing of the data is similar to that found in ".FTR" files. Here, there are q power points grouped together representing the power in each of q frequency bins for each channel. Power spectrum levels are typically stored as decibels (dB). Once again, different formats for the data storage are acceptable as long as they conform to what is described in the file's header, but the format of Figure 3 is by far the most common.

2.4 VAX and READRT File Formats

Files from TRANSFER are stored on VAX disks in one of two formats. The most common of these is a binary direct access file. This format is compact and allows direct access to any block in the file.

Data files are typically very large, often taking an entire 2400 foot reel of 9 track magnetic tape, so transferring them to disk causes storage problems to appear quickly if several people are doing analysis. For this reason the disk files should be stored in as compact a form as possible.

Random access to any part of the data in a file is also important. A scientist doing time series analysis may be interested in only certain segments of data in a large file, so having to sequentially access each record to get to the desired one would be inefficient. After analysis, display programs (such as PLTPWR) also need random access to the data so that any segment may be displayed rapidly and in any order. The format of the VAX files permits this.

The other format is called READRT after the file transfer program which originally used it. This format is used by some of the older analysis programs, and results in a sequential unformatted file. It does not take a great deal more disk space than the VAX format, but the advantages of random access to data are lost. It is not recommended that this file format be used in future analysis programs.

3. PROGRAM IMPLEMENTATION

This section begins with a description of the program structure and the features available in TRANSFER. Implementation details are then presented, and non-standard practises are discussed more fully.

TRANSFER was written in FORTRAN 77, but is not easily transportable to computers other than VAX and micro-VAX models made by Digital Equipment Corporation due to its extensive use of system calls. (System calls use internals of the VMS operating system directly and are not part of the FORTRAN 77 standard.) The system calls were used to speed up tape and disk access, and to make use of some of the powerful capabilities of the VAX-VMS operating system. On the bright side, however, the program can be run on any of the VAX machines from DEC without modification. TRANSFER can be used from any ANSII standard computer terminal, but works best with a VT100/200 series terminal or emulator.

Subroutines from many sources were used in TRANSFER. Asynchronous disk input/output routines from NRL (the Naval Research Laboratory) in Washington D.C. proved to be very useful in this implementation [3]. Other useful routines from various groups at DREA have been incorporated to avoid duplication of programming effort. Subroutines obtained from outside sources will be noted as such in the following discussion.

3.1 Program Structure

TRANSFER was written in a modular format in order to facilitate modifications and additions. 'User-friendliness' and simplicity of use were major considerations in program design. Flexibility is a keyword for TRANSFER since many input file formats must be accessible to VAX users, and the data in those files should be easy for the user to manipulate. Simplicity of use and a high degree of flexibility are not always compatible, but the attempt has been made to achieve both objectives with TRANSFER.

Even the most efficiently written program can be practically useless if it has a poor user interface. For this reason, considerable effort was put into making the user interface of TRANSFER easy to understand *and* use. The terminal input session has been separated into related modules (for example, one module deals with defining the section of an input file to be transferred into the output file). Each module is presented on a separate screen on the user's terminal, and a heading appears at the top of the screen describing the purpose of the module. Examples of this will be given in a later section.

Program structure is outlined in the flowchart of Figure 4. There are three basic segments - input, processing, and output. Within these segments, subroutine structures were used when possible. These are not noted in the chart, but more detail on some of them will be given in a later section. The loop structure of the program is fairly simple at the flowchart level but became rather difficult to implement because of the differences in input file structures which had to be accommodated. Most of the options available to the user are noted in the chart, and will be discussed in detail in the following sub-section.

3.2 Program Features

Input files for TRANSFER can be located either on magnetic tape or on disk. In the case of tape, ".DAT", ".FTR" and ".PWR" files are accepted, while only VAX format disk input files are accepted. One of the inconveniences of using magnetic tapes is the requirement that the user must remember to **allocate** the tape unit and **mount** the tape (both VMS commands). TRANSFER avoids this by the use of VMS system calls. When the user specifies a tape unit (for example MSB0:) as part of an input filename, TRANSFER tries to allocate that unit and then mount any tape found on it. If the unit is already assigned or the tape cannot be mounted, the user is informed and program execution halts; otherwise the operation is transparent.

The program will not stop executing if it encounters a parity error while reading from a tape file. Parity errors on ".DAT" tape files are a distinct possibility due to the manner in which the PDP-11 data collection programs operate. High speed is the priority for the data collection programs, so no error checking is performed while writing to tape (resulting in the possibility of parity errors). (Error checking uses valuable time and could cause data to be lost in some cases.) When TRANSFER hits a parity error, it rewinds the tape to the last good record of data and substitutes that for the corrupted data. This action could bias the statistics of the output if many parity errors are encountered, but was deemed to be more appropriate than destroying the time synchronization of the file by throwing away data. The user is notified each time a tape parity error is encountered, and a running count of the errors is presented.

TRANSFER allows wildcards to be used to specify a family of input files with similar names (or parts of names) for input from *either* disk or tape. A short description of wildcarding would be appropriate for the uninitiated - it hinges upon the use of a wildcard character (in this case '*'). Filenames are made up of a *name* and an *extension* separated by a period (for example TEST.DAT). The family of files with the name TEST and any

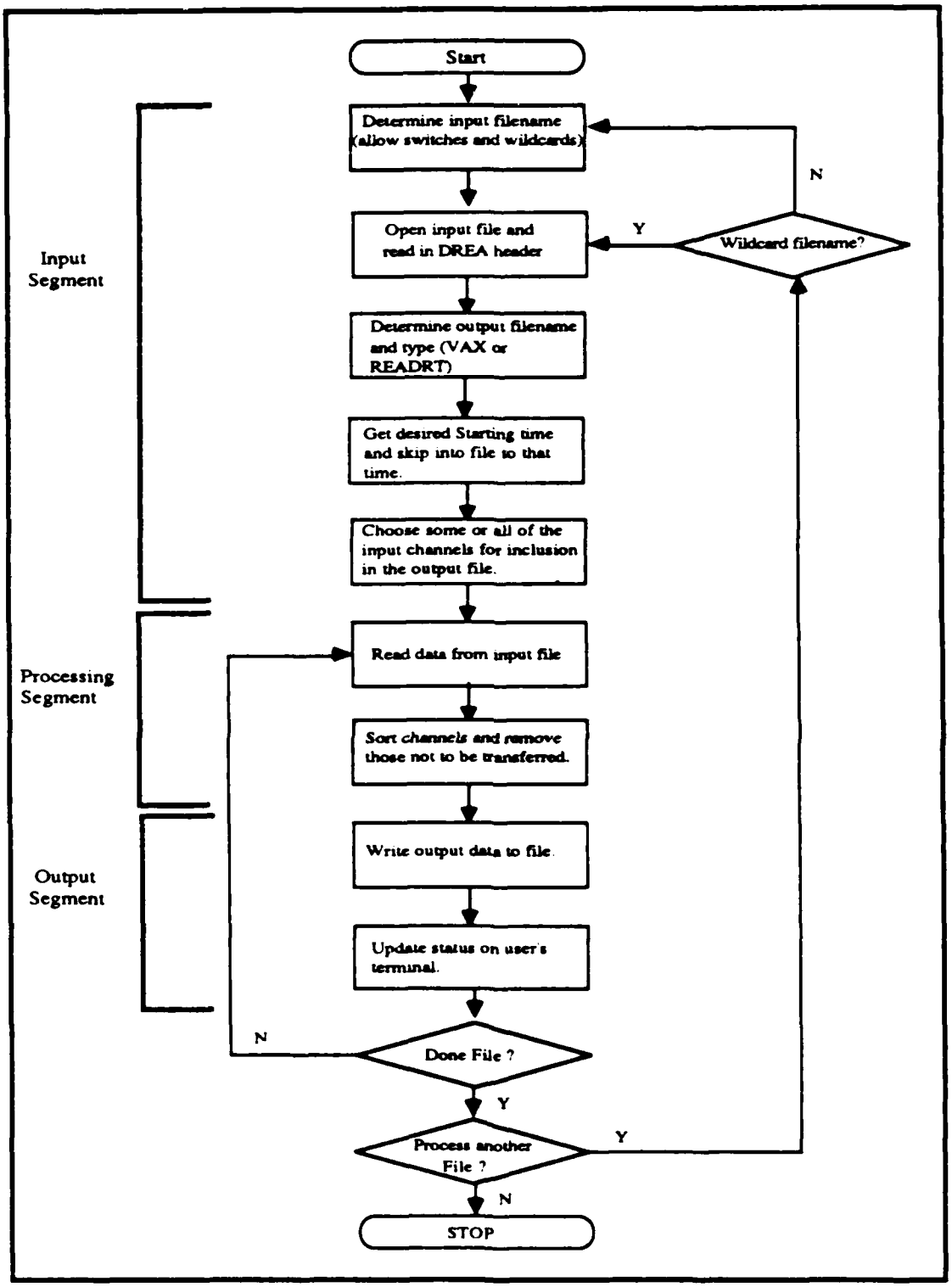


Figure 4: General flowchart for TRANSFER

extension can be specified by typing TEST.*. Similarly all files with the extension ".DAT" can be specified by typing *.DAT. Typing TES*.DAT would specify all files having a name beginning with TES and having the extension ".DAT". When TRANSFER encounters a wildcard in a filename, it finds all files which satisfy the input name set and processes them in sequence. A user can specify a new set of transfer parameters for each input file, or alternatively can set up parameters for only the first file and use those same parameters for all of the other files.

Tape files present other possibilities as well. A user may wish to transfer a number of files from tape without having to specify their names (for example the second through the fifth files on the tape). This option is available to TRANSFER users through an input file switch (switches are only available when using input files on magnetic tape). A switch is used in the following manner: the user types the name of the tape unit upon which the reel is mounted, followed by a slash (/) and a switch parameter. For example, to transfer the first through the fifth files from a tape on unit MSB0:, the user would type **MSB0:/START=1/STOP=5** when prompted to enter an input filename. Here the START switch defines the file on tape with which to begin the transfer, and the STOP switch defines the number of the final file which is to be transferred.

There is only one other switch available; that is the /V (or verify) switch. When this switch is included after an input tape filename (for example **MSB0:TEST.DAT/V**), TRANSFER will ask the user for verification before skipping any file which it encounters on the tape (If the first file found is the one the user specified, it is processed without question). This switch is useful if the user wishes to process the first file on a tape but doesn't know its name. In that case, when TRANSFER asks whether it should skip the file, the user need only give a negative reply and it will be processed. When /V is not used, the entire tape will be scanned for a filename match and no option for processing non-matching files will be presented to the user.

The default filename for files produced by TRANSFER is the same as the input filename with the extension .TFR to indicate that it is output from TRANSFER. The user can supply a different filename and/or extension if desired. Output file format (VAX or READRT) is also selectable.

The case will often arise where only a small portion of an input data file will be used for analysis or display. For this reason, TRANSFER allows the user to choose a segment of the input file for transfer. A user-selected segment is always defined by its start time. The time at which recording for a file began is included in the DREA header. This time is displayed for the user, who can then specify the time at which the desired segment of data begins. TRANSFER will then skip into the file to the desired time (a rather complex process, since the time and number of records to be skipped depend upon the type of file being transferred[†]). When the file has been positioned to the desired start-time, the user is given the option of specifying the segment length as a number of blocks (512 bytes/block), a time duration, or the remainder of the file. Thus a user has great control over the data transferred to the output disk file.

[†]For example, .FTR and .PWR files have a time resolution which is determined by the FFT length used in producing the file. For 2kHz samples and an 8K FFT, each set of Fourier coefficients covers a 4 s time interval. Due to the construction of the files, a time resolution of less than 4 s would be impossible in this case. In this situation, the actual start time is the accessible time closest to (but not less than) that requested by the user.

Full control over the channels to be put into the output file is also a necessity. Channel numbers which were used in data recording are entered in the byte section of the DREA header and so are available during TRANSFER operation. The user can select any subset (or all) of the channels in the original input file for transfer into a VAX file. The header of the new file thus created will be modified to include only channel numbers of those channels presently in the file. This capability is useful in reducing the amount of data stored on disk since non-acoustic channels or channels known to contain corrupted data need not be transferred to disk.

During operation, TRANSFER provides feedback to a status screen on the user's terminal. This screen gives information on the parameters set up for the file transfer and on the progress of the transfer.

3.3 Implementation Details

This section presents a more detailed look at the structure of TRANSFER. Each major segment of the program is shown in a flowchart indicating which operations are carried out in the main program, and which of them are carried out in subroutines.

Figure 5 shows the file selection segment of the program. Wildcarding and switches are implemented in this segment. The user is first prompted for a filename, and then the name is processed to check for switches. If switches are present, they are decoded and the proper flags set for later processing. If a wildcard character appears, a flag is set for tape processing, or a check of the appropriate directory is performed and all matching names extracted for disk file processing.

Once the input name has been processed, the first file which matches all criteria is found and opened (not necessarily an easy task when using magnetic tape). If the file opening was successful, the DREA header is read into a buffer for use in setting up the transfer parameters. System calls are used for all of the magnetic tape operations in the interests of speed. Disk operations are done using a set of subroutines obtained from the Naval Research Laboratory. These allow asynchronous operations (ie. computations can carry on while data are being read from the input file) and are written in VAX-MACRO, so they offer a speed advantage over pure FORTRAN calls.

When a file has been properly opened, the user is prompted for an output filename. A default (described in the preceding section) is presented, but this can be changed to anything the user wants. If wildcard files are used, the user is given the option of specifying transfer parameters separately for each file or of using the transfer parameters set for the first file for all of the others. Output format (VAX or READRT) is then chosen, and the output file is opened.

The flowchart shown in Figure 6 gives more detail of the structure in the processing and data output sections of TRANSFER. The information contained in the DREA header is used heavily in this segment of the program. Parameters such as record size, the multiplexing type and the number of channels are used to determine the the number of blocks which must be skipped to get to the desired point in the input file. TRANSFER works with "frames" of data. A "frame" is defined as the smallest amount of data which can be read from the input file which gives a full set of input data for each channel in the file. A full set of data can vary from a single time sample in a ".DAT" file to a number of points defined by the FFT length used for processing in a ".FTR" or ".PWR" file. This length is also determined by the type of multiplexing used in the file (see Figures 1, 2 and 3). As well, a "frame" defines the minimum time unit which can be accessed by TRANSFER.

The subroutine which is used to skip to the desired time in the input file and to define the segment length for transfer makes extensive use of VMS system calls when tape is being used as the input medium. Because of the random access nature of VAX format disk files, picking the correct starting time is as simple as addressing the proper block when the input file is on disk. Some subroutines written by D. Peters and L. Bunch (both DREA summer research assistants) were incorporated in this routine to improve the user interface.

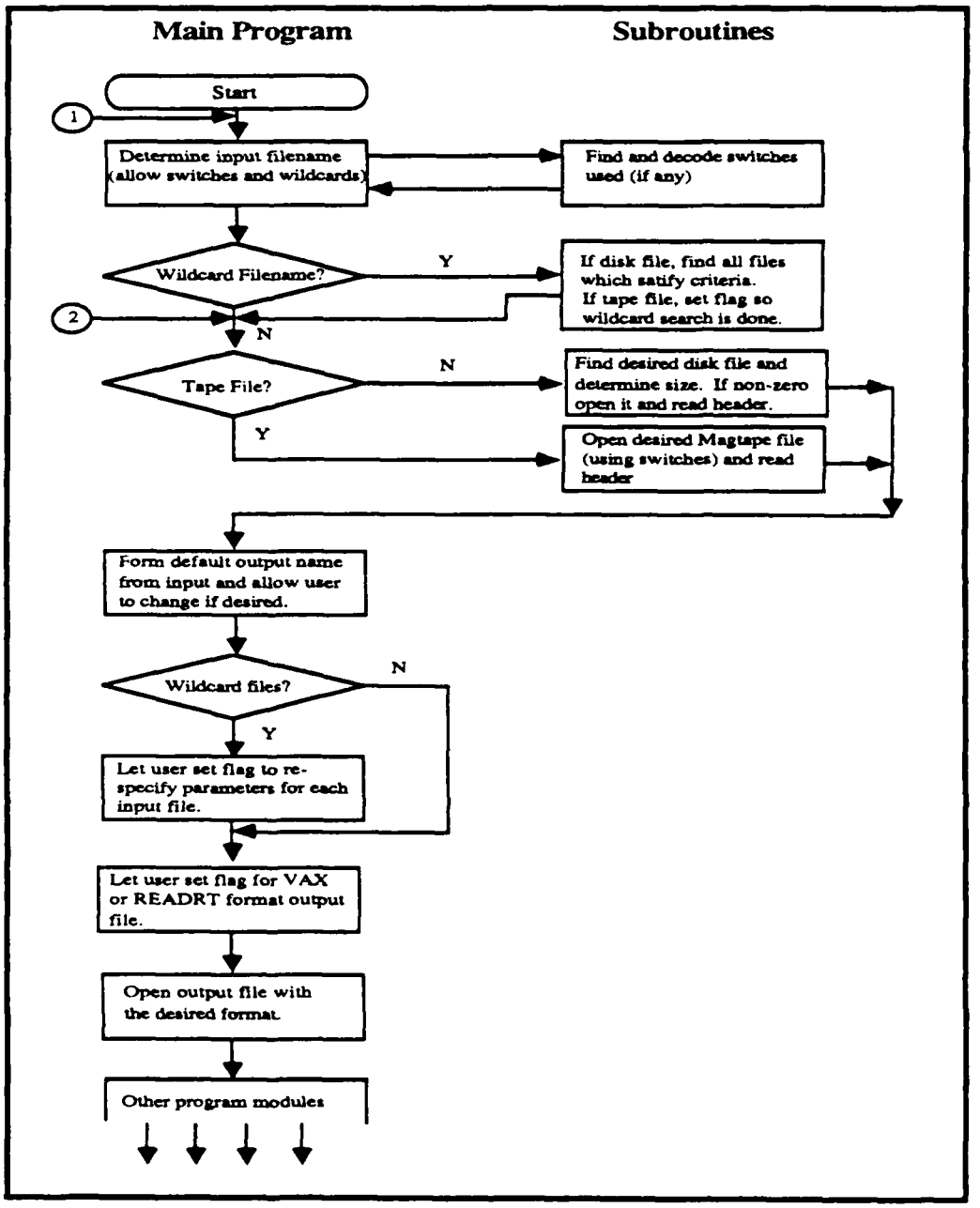


Figure 5: Details of File Selection Segment of TRANSFER

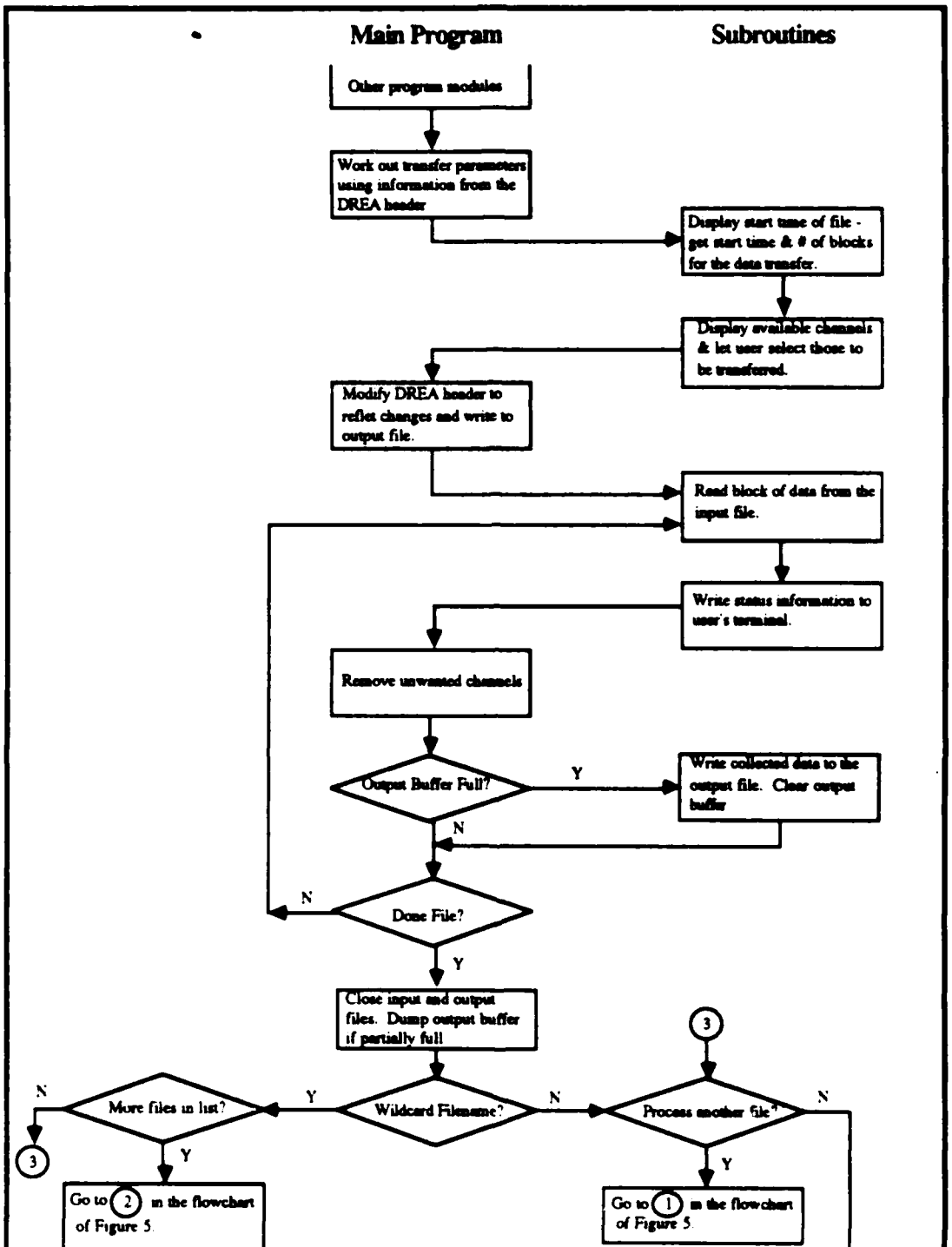


Figure 6: Details of processing in TRANSFER

Determining which channels are to be processed is simple enough if a single file is being used as input. If a wildcard file set has been chosen, the process becomes more difficult. Channels can be selected by "number", that is by their standing in the order in which they were put into the file (first, second, etc. in the multiplexing hierarchy), or by the hydrophone channel which they represent (included in the byte label of the DREA header). For example, the "first" channel recorded in a ".DAT" file may actually correspond to the time series for hydrophone channel #20. The user has the option of using the same parameters for each input file in a wildcard set, and so must decide whether to key on channel numbers or hydrophone numbers. While such a set is being processed, it is possible that channels will occur in a different order in some of the files. The headers of the output files will reflect this, but it is often better to key on hydrophone numbers in such situations. If a chosen hydrophone does not exist in one of the input files, the user is notified during execution and allowed to change the selection.

Once channels have been selected, the DREA header is modified accordingly and written into the output file. At this point the data transfer can begin. Enough blocks are read from the input file to give at least one "frame" of data. If some channels are not being transferred, these are then removed from the input buffer. It is most efficient to write large segments of data to disk, so input blocks are processed until a relatively large output buffer is filled or the input file is finished. The collected (perhaps reduced) data are then written to the output disk file using the DBIO routines from NRL (for a VAX format file) or FORTRAN write statements (for a READRT file).

Once a file is completed, input and output files are closed before proceeding. If wildcard files have been selected, a check is made for further matches and if any are found, the next matching file is processed. (The user may or may not be prompted for transfer parameters for files after the first, depending on the option chosen.) If wildcard files are not being processed, or no more wildcard matches are found, the user is asked if more files are to be processed. At this point, the program can be exited or a new run started.

4. HOW TO USE THE PROGRAM

A sample terminal session will be presented in this section. "Snapshots" of the user's terminal screen will be used to illustrate the user interface and provide information on running the program.

The executable version of TRANSFER is located in the directory SAS:[FARRELL.TRANSFER] (soon to be moved to DREAPACS:[TRANSFER]). To start the program, the user must type **RUN SAS:[FARRELL.TRANSFER]TRANSFER.**

The following scenario is presented as an example. A user has a mag-tape (produced by a PDP-11 data recording program) containing time series data which are to be transferred to the VAX. He believes the filename to be ZZZ003.DAT, but is unsure of the extension. He does, however, know that the file begins at 10:33:41 and that data from 32 channels are recorded in the file. Ten seconds of data beginning at 10:33:50 and 4 channels out of 32 in the data file, namely 2, 11, 18 and 23 are to be transferred. TRANSFER is started using the command mentioned in the preceding paragraph, and then an input session begins.

In the following figures, a header categorizing the parameters to be input appears at the top of each input screen (boldface type). After every prompt, the default value is presented in brackets. To accept the default value, the user need only enter a carriage return. Help can be obtained after any prompt by typing a "?" followed by a carriage

return. User-entered responses appear in boldface type (after a prompt, so there should be no confusion with the screen headers). Some prompts appear only under special circumstances (when a wildcard file input set is being used, for example), and these are shown in italics when they would not otherwise appear in the example being presented.

The first screen which appears once TRANSFER begins execution is shown in Figure 7. Here the user sets up the input and output file specifications.

File Setup

Enter name for the input datafile. (MSB0:Q38334.DAT?) ? **?**↵
File name: UNIT or STRUCTURE:NAME.EXTENSION
Enter name for the input datafile. (MSB0:Q38334.DAT ?) ? **MSA0:DUMMY.DAT/V**↵

%MOUNT-I-WRITELOCK, volume is write locked
%MOUNT-I-MOUNTED, RT11A mounted on _MSA0:

Found file: ZZZ003.DAT
Want to pass over this file?(-1=y/0=only/1=no/2=rewind) (-1?) ? **1**↵

Default output file extension is T FR - Change it? (N?) ?
Enter the new default file extensi on. (TFR?) ?
Re-specify output parameters for e ach input file? (N?) ?

Enter a name for the Disk output file. (ZZZ003.TFR?) ?↵
Should the output file be Vax (V) or READRT (R) format? (V?) ? **V**↵

NOTE: ↵ signifies a carriage return.

Figure 7: First Input Screen of TRANSFER

To demonstrate the use of the "help" feature, a "?" was entered in response to the prompt asking for an input filename. The help field shows that the name format is UNIT:NAME.EXTENSION, and the prompt is then repeated. The unit or directory must be the first entry in the file specification - if no unit or directory is specified, the current directory on disk is used as a default.. In the example, the tape containing the data is mounted on unit **MSA0:**. The user doesn't know the name of the file to be transferred, so a dummy name has been entered along with the /V switch (this means that the program will ask before skipping any file). Messages from the system following the filename entry show that the tape has been successfully mounted on the requested unit and that the tape is write locked (ie no data can be written to it - the safest policy with data tapes).

The /V switch causes the name of the first file found (ZZZ003.DAT) to be displayed. The user then has several options. If a -1 is entered, the file will be skipped and the search for DUMMY.DAT will continue. Entering 0 will cause ZZZ003.DAT to be skipped and the next file found to be processed. Entering 1 will cause ZZZ003.DAT to be chosen as the input file, and entering 2 will rewind the tape and begin searching from the start of the tape again. The default is to skip the file, but in this case, the user has chosen to process it by entering a **1**.

Next, a name for the output disk file must be specified. The default filename is the same as the input filename with the extension .TFR, and in this case, the user has chosen to

go with the default by entering only a carriage return. (If wildcard input filespecs had been used, the user would have been prompted to determine if the .TFR file extension should be used for all output files. At that point, a new extension could have been specified which would appear on all output files from the wildcard set. The option to re-specify output parameters for each input file would also have been presented if a wildcard set had been chosen.) Finally, the output file type must be chosen. Here, the user has chosen VAX format (described earlier) and entered a V followed by a carriage return (a carriage return would have been enough since V is the default, but entering V does no harm).

The next screen to appear deals with selecting a file segment to be transferred and is shown in Figure 8. At the top of the screen, the start time of the chosen file is displayed. The user is then given the option to start the transfer at a later time in the file (default is to start at the beginning). Here, the user asks to start the transfer at 10:33:50. TRANSFER skips into the file to the desired point (or the nearest accessible time greater than that requested - constrained by the input file format). The user is notified of the start time which the program is actually using and of the number of blocks being skipped. The next prompt allows the user to set the amount of data to be transferred. The user has three options; n (where n is some number) will transfer n 512 byte blocks (to the nearest "frame") into the output file; -1 will transfer all data from the specified starting position (time) to the end of the input file into the output file; and -2 will allow the user to specify a time interval for transfer. In this example, the user has chosen to transfer by time interval. Time is specified in the HH:MM:SS format, and in the example, the user has asked for 10 seconds worth of data to be put into the output file. (If a wildcard file set had been specified for input, the user would, at this time, be asked to decide whether to transfer the same channel set or hydrophone set for each input-output file pair. This choice doesn't deal with "data segment specification", but was placed on this screen due to program structure constraints. The prompt appears in italics as wildcards are not being used here.)

Data Segment Specification

This file starts at 10:33:41

Do you want to begin processing at some other time (N?) ? **Y** ↵

Enter the time at which you wish to start (10:33:41?) ? **10:33:50** ↵

Actual start time will be 10:33:50
144 physical blocks will be skipped.

Enter n to X-fer n blocks,-1 for all,-2 to specify time (-1?) ? **-2** ↵

Enter the length of time of the transfer (00:01:00?) ? **00:00:10** ↵

Select the same H P (Y) or channels (N) from each file ? (Y?) ?

Figure 8: Second Input Screen for TRANSFER

Once a file segment has been defined, a channel set-up screen appears (shown in Figure 9). The total number of channels in the input file (including non-acoustic channels) is displayed, followed by a list of the acoustic channels and their corresponding hydrophone numbers (from the byte part of the DREA header). From these, the user

selects which channels are to be transferred into the output file. The number of channels to be transferred is specified first (-1 will transfer all channels - including non-acoustic ones - to the output file), and then the channels (not hydrophone numbers) are selected. The selected channels can be separated by spaces or commas. This concludes the input session as TRANSFER has all the parameters needed for execution.

Channel Setup

There are 32 channels in the input file.

The following are acoustic channels:
1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,
27,28,29,30,31,32

The corresponding H/P numbers are:
1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,
27,28,29,30,31,32

How many channels do you want to process? (-1 for all) (-1 ?) ?**4** ↵

Enter the channels you wish to study : **2 11 18 23** ↵

Figure 9: Third Input Screen for TRANSFER

Once data transfer has begun, program status is continuously updated on the terminal screen. The status screen (shown in Figure 10) can be divided into three segments. (Note that parameters which depend on choices the user makes during the input phase of the program appear in boldface type in the Figure.) The first of these segments displays the current time and date and is updated as the program executes.

The second segment is static and gives information on the input and output files. Part of the ASCII label is shown to give some indication of the origins of the file. Below this label, the names and start times for the input and output files are displayed. The number of frames requested for transfer to the output file is also shown. If all blocks from the specified start time to the end of the input file are to be transferred, the number of frames to be transferred is shown as "TO EOF" which stands for To End Of File (Note that this is shown in the sample screen of Figure 10 - which came from a different run of TRANSFER with the same input file). The total number (including non-acoustic) of channels in the input file and the number being transferred are shown, along with their channel numbers and hydrophone numbers (acoustic channels).

The third segment is dynamic and updates as the program executes. The number of frames processed is shown along with the average time taken to sort and transfer each frame to the output file. In the sample screen shown, 100 blocks have been processed, and the average time (clock time) per frame overall has been .02 seconds. A running count of parity errors during tape reads, is given below the "average time per frame" block if any errors occur.

When all data have been transferred to the output file, the user can request that another file be transferred. In this case, the program starts again with the first input screen. If a wildcard set was specified as input and the same transfer parameters were to be used for each file, the next transfer will proceed automatically. If transfer parameters were to be

specified for each file in a wildcard set, the user goes through all input screens except the first for each file. If a tape file was used as input, the tape will be dismounted and rewound automatically when execution of the program finishes.

FILE TRANSFER STATISTICS
4-DEC-1986 11:50:00

ASCII Label: **32 CHANNELS REAL**

Input File: **MSA0:ZZZ003.DAT** Output File: **ZZZ003.TFR**

Starts at: **10:33:41** Starts at: **10:33:50**

Number of frames requested: **TO EOF**

Channel usage: **4** chosen out of **32** total

Channels: **2,11,18,23**

Acoustic Channels: **2,11,18,23**

Number of Blocks processed: **100**

Average time per frame: **.02** sec

Figure 10: Program Status Screen

Appendix A gives a list of all prompts which can appear during transfer execution, along with the 'help' string for each. As well, full descriptions of prompts which have not been discussed fully in the text are given. Messages which appear on the screen when an event requiring user attention occurs during TRANSFER execution are also listed and explained in the Appendix.

For completeness, a listing of TRANSFER and its major subroutines is included in Appendix B. Some of the VMS system calls are unavoidably confusing; however the programs are fully commented, so no further description will be given in the main body of this document.

5 FUTURE DEVELOPMENTS

Currently, output files cannot be written to magnetic tape. This feature would be useful, since no tool for reducing the size of data files exists on the PDP-11s (PDP-11 users have the same need for this functionality as VAX users). Squeezing several ".DAT" files onto a single mag-tape would be another use for a tape-output feature (no tool exists for doing this on the VAXen at present). Such a modification requires a module for PDP-11 format tape output (currently being developed by G. Heard at DREA). Once the module is obtained, the upgrade should take relatively little time.

Transfer of other file formats would also be desirable (for example READRT to VAX conversion), and will be implemented if enough user interest is shown. A generic

transfer option (block-for-block copy of an input mag-tape file to VAX disk) is being developed and should prove useful for non-DREA generated tapes.

6 CONCLUSIONS AND ACKNOWLEDGEMENTS

This note has described a robust file transfer program for the VAX computer. The program allows transfer of PDP-11 format magnetic tape files (and VAX disk files) to VAX disk files which are formatted properly for use by the Surveillance Acoustics section suite of signal processing and display programs. Data manipulation tools which allow selected channels and data segments to be transferred are available within the program, making it a versatile tool for pre-analysis data preparation.

Thanks to Ed Chaulk, Vance Crowe, Phil Staal and many others who made useful suggestions during TRANSFER's creation. Almost all of their suggestions have been incorporated in the working program. Asynchronous disk input/output routines written by J. Padgett of NRL and obtained with the help of Art Collier at DREA proved to be extremely useful, increasing the speed of TRANSFER significantly. Subroutines written by Vance Crowe, Laurie Bunch and Doug Peters were also used, making my programming job a great deal easier. Figures 1, 2 and 3 were produced by Phil Staal. Thanks should also go to Bruce Skinner for his aid with VAX VMS system calls.

References

1. **Caldwell D.A.,** "A Standard for the DREA Data Descriptor Block", D.R.E.A. Technical Communication 87/302 , Dartmouth, Nova Scotia, December 1986.

2. **Farrell J.B.,** "SAFTR - A Program for Producing Fourier Coefficient Files on the VAX Computer", D.R.E.A. Technical Communication 87/ (DRAFT) , Dartmouth, Nova Scotia, February 1987.

3. **Hurdle B.B.,** "Private Communication", Naval Research Laboratory, Washington, D.C., February, 1986.

A - DETAILS OF USER PROMPTS - INTRODUCTION

Appendix A1 lists all of the prompts which can appear on the user's screen during TRANSFER execution and gives a brief description of each. The help available for each prompt is also listed. Appendix A2 lists messages which appear on the screen when an event occurs during TRANSFER execution which requires attention or should be noted by the user.

A1 - Alphabetical Listing of Prompts

PROMPT: Default output file extension is .TFR - change it?
HELP: All output files will have the specified extension.
DETAILS: Allows the user to change the default file extension which will be used when processing wildcard input file sets. If parameters are not being changed for each file in the set, output filenames will be the same as the input filenames, but with the default extension substituted for the original.

PROMPT: Do you want to begin processing at some other time?
HELP: Default is to start at the time shown.
DETAILS: Appears when the start time of the input file is displayed. The user can start the output file at the same time, or modify the start time for output by responding "Y" to this prompt.

PROMPT: Do you want to process another file?
HELP: Default is to exit the program.
DETAILS: Allows processing to continue when the current input file (or set) has been completed.

PROMPT: Enter a name for the disk output file.
HELP: Default will be the same as the input name with the extension .TFR.
DETAILS: Defines a file where the output will be dumped.

PROMPT: Enter n to X-fer n blocks,-1 for all,-2 to specify time.
HELP: -2 will let you enter a time interval for the transfer.
DETAILS: Defines the segment of data to be transferred. Data will start at the specified start time and have an extent specified by the response to this prompt.

PROMPT: Enter name for the input data file.
HELP: File name: UNIT or STRUCTURE:NAME.EXTENSION
DETAILS: Defines a file (can be a wildcard file set and have switches included in the filename) where the input data are to be found.

PROMPT: Enter the channels you wish to study.
HELP: No help appears for this prompt.
DETAILS: Allows input of a vector of channel #s which are to be transferred. This prompt appears after the number of channels to transfer has been established and a list of the available channels has been displayed.

PROMPT: Enter the length of time of the transfer.
HELP: Format is HH:MM:SS.
DETAILS: The user must enter the time extent of the data segment to be transferred from the input to the output file.

PROMPT: Enter the new default file extension.
HELP: Typically three letters long (leave out the ".").
DETAILS: Allows a new default file extension to be set. Appears after the prompt allowing the user to decide whether or not to keep the .TFR extension.

PROMPT: Enter the time at which you wish to start.
HELP: Format is HH:MM:SS.
DETAILS: Set a new start time for the output file. The prompt appears after the user requests a transfer start time other than the start time of the input file.

PROMPT: How many channels do you want to process? (-1 for all)
HELP: Enter the number of channels to process.
DETAILS: Lets the user select from the channels available in the input file.

PROMPT: Proceed using the subset of requested phones found?
HELP: Re-specify H/P or skip this file if the reply is N.
DETAILS: Prompt appears when the required H/P set is not found in a wildcard file. Execution can continue with the subset of H/P found in the file. Alternatively, the user can modify the H/P set or skip the file and proceed to the next one in the set.

PROMPT: Re-specify output parameters for each input file?
HELP: Else use the default output file name & specs.
DETAILS: An option used for wildcard input file sets. If the user replies "N" the parameters entered for the first file in the set will be used for all of the others, and the output filenames will be the input names with the default extension. If the user replies "Y" all prompts will appear for each input file in the set.

PROMPT: Select the same H/P (Y) or channels (N) from each file?
HELP: Default will select the same H/P from each input file.
DETAILS: Used in conjunction with wildcard file sets when transfer parameters are not being changed for each file in the set. Either H/P or channels will be kept the same for each output file.

PROMPT: Should the output be VAX (V) or READRT (R) format?
HELP: (V) format compatible with DISPVAX, SASPEC, etc.
DETAILS: Sets the format of the output data file.

PROMPT: Skip to the next file in the input set?
HELP: Otherwise use this file with reduced # of chans.
DETAILS: Prompt appears (when channels are being kept the same in a wildcard set) when channels differ from those expected in the file set. The user can proceed with a transfer of the reduced number of channels, or skip to the next file.

PROMPT: Type 1 to take closest record start, 2 to re-specify time.
HELP: Closest may be earlier or later than the chosen time.
DETAILS: Appears when the user has requested the transfer of a number of blocks which doesn't result in an integral number of data frames in the output file.

PROMPT: Want to pass over this file?(-1=y/0=only/1=no/2=rewind)
HELP: -1=pass file&look at next/0=correct/1=use file/2=rewind.
DETAILS: Used in conjunction with the /V switch for skipping files on mag-tape. The user can: (-1) continue searching for an exact match for the input filename, (0) skip to the next file and process it, (1) process the file found or (2) rewind the tape and then continue the search.

A2 - User Messages From TRANSFER (Grouped By Subroutine)

A2.1 Messages from TRANSFER main program

MESSAGE: Stop *** fatal -- input file is empty.
SOURCE: TRANSFER main program.
CAUSE: Appears when the user tries to open an input disk file which contains no data.

MESSAGE: Dismounting Tape.
SOURCE: TRANSFER main program.
CAUSE: Tape input was completed or end of tape was reached.

A2.2 Messages from subroutine Check

MESSAGE: Error in system call at position ***.
SOURCE: Subroutine CHECK.
CAUSE: An error condition resulted during a VMS system call.

A2.3 Messages from subroutine MTFILE

MESSAGE: Stop *** All files processed.
SOURCE: Subroutine MTFILE.
CAUSE: All files from a mag-tape /START /STOP set have been processed.

MESSAGE: End of tape encountered - rewinding.
SOURCE: Subroutine MTFILE.
CAUSE: Reached the end of a mag-tape being used for input.

MESSAGE: Skipping to next file.
SOURCE: Subroutine MTFILE.
CAUSE: A file with a name different than the input filename was found and is being skipped (either because the user requested it or because the /V switch was not used).

MESSAGE: File **** found.
SOURCE: Subroutine MTFILE.
CAUSE: Found a file which matches the input filespec.

MESSAGE: Found file ****.
SOURCE: Subroutine MTFILE.
CAUSE: Appears for every file found when the /V switch is used in the input filespec.

MESSAGE: Assume you want to skip to the next file.
SOURCE: Subroutine MTFILE.
CAUSE: Appears when the user responds ambiguously to the prompt asking whether or not to skip the file found.

A2.4 Messages from subroutine GETTAP

MESSAGE: Tape is already mounted - assuming you did it.
SOURCE: Subroutine GETTAP.
CAUSE: A mount request has been issued for a tape unit that was already mounted.

MESSAGE: Error - device probably allocated to another user.
SOURCE: Subroutine GETTAP.
CAUSE: Failure in a request to allocate a tape unit to the TRANSFER job.

MESSAGE: Device already allocated to you.
SOURCE: Subroutine GETTAP.
CAUSE: Routine tried to allocate a tape unit already allocated to the user.

A2.5 Messages from subroutine Channel_Select

MESSAGE: Channel Select - no acoustic data found in file.
SOURCE: Subroutine Channel_Select.
CAUSE: The byte label of the DREA header indicated that there were no acoustic channels present in the file.

MESSAGE: Working on file ***.
SOURCE: Subroutine Channel_Select.
CAUSE: Informs the user of the current file being processed in a wildcard file set.

MESSAGE: The H/P available differ from the originals.
SOURCE: Subroutine Channel_Select.
CAUSE: One or more of the requested hydrophone channels is not present in the wildcard file being processed.

MESSAGE: You are keying on channels rather than phones, so I am proceeding.
SOURCE: Subroutine Channel_select.
CAUSE: Hydrophone numbers have changed in a wildcard file-set but the user is keying on channels rather than phones, so this serves as a warning.

MESSAGE: You asked for a channel not found in the input file.
SOURCE: Subroutine Channel_Select.
CAUSE: User is keying on channels, and one (or more) of the required ones was not found in the current wildcard input file.

MESSAGE: Proceeding with reduced # of channels.
SOURCE: Subroutine Channel_Select.
CAUSE: User has chosen to process a file, even though it doesn't contain all of the files originally asked for.

MESSAGE: There are ## channels in the input file.
SOURCE: Subroutine Channel_Select.
CAUSE: Informs the user of the number of channels available.

MESSAGE: The following are acoustic channels.
SOURCE: Subroutine Channel_Select.
CAUSE: Informs the user of the acoustic channels in the input file.

MESSAGE: The corresponding H/P numbers are:
SOURCE: Subroutine Channel_Select.
CAUSE: Informs the user of hydrophone numbers corresponding to channel numbers.

MESSAGE: Channel Select - sorry no default.
SOURCE: Subroutine Channel_Select.
CAUSE: The user selected no channels for transfer to the output file.

A2.6 Messages from subroutine Skipper

MESSAGE: This file starts at : HH:MM:SS.
SOURCE: Subroutine Skipper.
CAUSE: Informs the user of the start time of the current input file.

MESSAGE: Actual start time will be HH:MM:SS.
SOURCE: Subroutine Skipper.
CAUSE: Informs the user of the actual start time of the output data file. May be different from the requested time due to data frame size in the input file.

MESSAGE: Disk start block will be: ###.
SOURCE: Subroutine Skipper.
CAUSE: Informs the user of the start block number of the input disk transfer.

MESSAGE: ### physical blocks will be skipped.
SOURCE: Subroutine Skipper.
CAUSE: Informs the user of the blocks to be skipped on tape.

MESSAGE: You specified a zero-length transfer - try again.
SOURCE: Subroutine Skipper.
CAUSE: User specified a transfer time less than the frame time of the input file.

MESSAGE: WARNING - you are trying to transfer 0 blocks - try again.
SOURCE: Subroutine Skipper.
CAUSE: User specified a number of blocks less than the number of blocks in a data frame.

MESSAGE: Transfer time must be at least: HH:MM:SS.
SOURCE: Subroutine Skipper.
CAUSE: Informs the user of the minimum time which can be specified for a transfer.

MESSAGE: You are not using an integral number of records.
SOURCE: Subroutine Skipper.
CAUSE: User tried to transfer data which did not fit into an integral number of frames.

A2.7 Messages from subroutine Disk_Wildcard

MESSAGE: No more files match the input spec.
SOURCE: Subroutine Disk_Wildcard.
CAUSE: All files matching a wildcard specification have been found.

A2.8 Messages from subroutine Reader

MESSAGE: Error reading record!! Error count = ##.
SOURCE: Subroutine Reader.
CAUSE: Routine encountered a parity error on tape and successfully passed it.

MESSAGE: End of file encountered.
SOURCE: Subroutine Reader.
CAUSE: End of an input file was encountered during a read.

MESSAGE: Saving ### frames and exiting.
SOURCE: Subroutine Reader.
CAUSE: Informs the user how many data frames are being stored after an end of file was encountered during a read.

MESSAGE: Read puts us past EOF - blocks to read ###.
SOURCE: Subroutine Reader.
CAUSE: Disk read would go past EOF.

MESSAGE: New blocks to read - ###.
SOURCE: Subroutine Reader.
CAUSE: Informs user of the number of blocks which will be read from a disk file when a full read would go past the EOF.

MESSAGE: Error encountered on disk read - saving what I can and exiting.
SOURCE: Subroutine Reader.
CAUSE: Informs the user that an error occurred during a disk read.

B - PROGRAM LISTING - INTRODUCTION

This Appendix contains listings of the TRANSFER program and its major subroutines. The casual user is unlikely to find these listings useful, however those writing programs for accessing PDP-11 tape files should be able to glean some information. The programs are commented fully, and most operations other than system calls should be easy to interpret. Print size has been reduced to conserve space.

B1 - Listing of the TRANSFER Main Program

```
C*****************************************************************
C                       PROGRAM NAME: TRANSFER                   *
C*****************************************************************

C
C                          Written by:
C                        Joseph B. Farrell
C                              DREA
C                            21 Jan. 1986
C
C                     Latest revision:  7 Aug. 1986
C
C       This program accepts a filename,opens that file on an RT-11
C    mag tape (or VAX disk file in standard 'PDP11S' format) and reads in the
C    DREA standard header.  The user can then choose some or all of the input
C    channels to be written into an output disk file (in standard 'PDP11S'
C    format or in the old 'READRT' format).  The tape unit does not need to
C    be mounted or allocated before the program is run.
C       The program was written to replace and expand on the functionality of
C    the READRT program written by  Ken Hahn of ASP.
C
C   SUBROUTINE CALLS:
C                  1)  ERASE_SCREEN= Clears the screen on the user's terminal.
C                  2)  VTMESS      = Puts a message on the user's terminal.
C                  3)  INPUTS      = Reads a string from user's terminal.
C                  4)  SYS$ASSIGN  = Subroutine to assign a channel.
C                  5)  LIB$STOP    = Stops program execution on error.
C                  6)  SYS$QIOW    = Queued input-output from a channel.
C                  7)  CHECK       = Checks the status buffer after a QIO
C                  8)  GETYN       = Gets a Yes or No response from the user.
C                  9)  DBOPEN(C)   = Opens (creates) a disk file.
C                 10)  DBSIZE      = Determines the size of a disk file.
C                 11)  DBREAD      = Reads data from a disk file.
C                 12)  DBWAIT      = Waits for disk operation to complete.
C                 13)  DBMUTE      = Turns of error reporting from DBMT routines.
C                 14)  SKIPPER     = Skips into a file to a specified time.
C                 15)  Channel_select = Chooses channels to process.
C                 16)  Write_header = Writes header to the output file.
C                 17)  INPUTI      = Reads an integer from the user.
C                 18)  DBWRITE     = Writes data to a disk file.
C                 19)  DBCLOSE     = Closes disk files.
C                 20)  SET_CURSOR  = Moves the cursor to a specified location.
C
C*****************************************************************
C                              MAIN CODE
C*****************************************************************
      PROGRAM TRANSFER
C.....................................................................
C---   Parameter and internal variable declarations.
C
      IMPLICIT INTEGER*4 (a-z)

      INCLUDE '($dmtdef)'

      PARAMETER MSG=5
      PARAMETER   (SS$ENDOFTAPE='878'X)

      BYTE              blabel(128)             !Byte part of DREA header
      BYTE              store_blabel(128)       !Temp. store for byte label
      BYTE              EOT_search(10)          !Checks for end of tape
      BYTE              output_data(10240)      !Output data
      BYTE              raw_byte_data(50000)    !Raw input data
      BYTE              raw_header(512)         !Raw header data

      CHARACTER         alabel*320              !ASCII part of DREA header
      CHARACTER         default_extension*64    !Default file extension
      CHARACTER         chan*6                  !Used in mag tape channel
      CHARACTER         default*1               !User input default
      CHARACTER         ifile*64                !Input filename
      CHARACTER         newtime*8               !Start time for processing
      CHARACTER         oldtime*8               !Start time of input data
      CHARACTER         ofile*64                !Output filename
```

APPENDIX B

```
      CHARACTER       otype                   'Flag for output file type
      CHARACTER       tape_mark(10)           'Checks tape marks
      CHARACTER       tape_name(10)           'Header filename
      CHARACTER       wfile*64                'Working filename


      INTEGER*1       blocks_from_input       'Blocks to read from file
      INTEGER*1       bytes_per_block         'Determined by input file
      INTEGER*2       file_type               'Input filetype (.DAT, etc.)
      INTEGER*2       hydrophones(128)        'Hydrophone numbers
      INTEGER*1       ilabe(16)               'Integer part of DREA header
      INTEGER*2       input_channel           'Input channel for tape
      INTEGER*1       number_of_channels      '# channels in input file
      INTEGER*2       number_sorted           'Number of elements sorted
      INTEGER*2       process(128)            'Channels to be processed
      INTEGER*2       selector_mask(128)      'For picking bytes
      INTEGER*2       shift_flag              'Flag for channel removal
      INTEGER*2       start_file              'Start for numbered tape files
      INTEGER*2       stop_file               'Stop for numbered tape files
      INTEGER*2       text_iosb(4)            'Status buffer

      INTEGER*4       blocks_per_record       'Determined by input file
      INTEGER*4       blocks_to_read          '# blocks for input read
      INTEGER*4       blocks_to_write         '# blocks for output write
      INTEGER*4       bytes_to_read           '# bytes read before sort
      INTEGER*4       channels_to_process     '# channels to process
      INTEGER*4       consecutive_channel_bytes
      INTEGER*4       disk                    'Flag for disk file input
      INTEGER*4       disk_channel_out        'Output channel
      INTEGER*4       frame_size              '# blks or records per frame
      INTEGER*4       frames_to_read          'Input frames to read
      INTEGER*4       iblk                    'Block for disk reads
      INTEGER*4       ierror                  'Error from disk reads
      INTEGER*4       itime
      INTEGER*4       disk_input_channel      'Input channel for disk
      INTEGER*4       jump                    'Increment for sorting channels
      INTEGER*4       frames_stored           'Counter for frames stored
      INTEGER*4       jrec                    'Counter for frames processed
      INTEGER*4       mask(2)
      INTEGER*4       nblocks                 'Number of blocks in input file
      INTEGER*4       oblocks                 'Default # of output blocks
      INTEGER*4       offset                  'Used for channel sorting
      INTEGER*4       sys$dismou              'Dismounts a tape
      INTEGER*4       tblocks
      INTEGER*4       total_blocks

      LOGICAL         another                 'Process another file.
      LOGICAL         end_of_tape             'End of tape indicator
      LOGICAL         exit                    'Flag for output extension
      LOGICAL         correct_file            'File flag
      LOGICAL         getyn                   'User response function
      LOGICAL         outspec                 'Re-do output filespecs
      LOGICAL         skip                    'Skip file flag
      LOGICAL         verify                  'Verify files to be skipped
      LOGICAL         wildcard                'Wildcard in filename


      REAL            block_time
      REAL            blksr                   'Blocks to read
      REAL            blksw                   'Blocks to write
      REAL            blkst                   'Temporary storage
      REAL            flabe(8)                'Floating part of DREA header
      REAL            fractional_blocks
      REAL            frame_fraction
      REAL            frame_time
      REAL            nrec
      REAL            records_per_read
      REAL            samples_per_input_block 'Determined by input file
      REAL            time_interval

      equivalence   (tape_name(1),raw_header(5))
      equivalence   (ilabe(1),raw_header(1))
      equivalence   (EOT_search,tape_mark)
      equivalence   (flabe(1),raw_header(33))
      equivalence   (blabe(1),raw_header(65))
      equivalence   (alabe(1),raw_header(193))

      common /io_statistics/rec_nres
      common /filer/wfile,ofile
      common /data/ output_data
      common /raw/ raw_byte_data

      external      io$_skipfile,io$_readvblk,io$_skiprecord,io$_rewind

C     Data for user interface.

      data ifile/'MSB:[130334.DAT'
     *     itime = 1
     *     blocks = 1
     *     start_file = 1
     *     stop_file = 9999
```

APPENDIX B

```
        default_extension = 'TFR'
C.....
C-- Clear the screen and write a header message.
C
        call erase_screen(1,1)
        call set_cursor(2,1)
        call vtmess('re','c',' File Setup '                    'VT100/200 SPECIFIC
C.....
C---    Get a name for the input file.
C
C       call inputs(prompt(1),ifile,help(1))
        call str$upcase(ifile ifile)
C.....
C---    Determine what switches were used in the input file specification.
C
        call switches( ifile,verify,start_file,stop_file)
C.....
C---    Add a .DAT extension if none was supplied and the filename was not
C---    specified as a wildcard.
C
        wildcard = .FALSE.
        if(index(ifile,'*').ne.0)wildcard = .TRUE.
        if(index(ifile,'.').eq.0)then
           i = len2(ifile)
           if(ifile(i:i).ne.'*')then
              ifile = ifile(:len2(ifile))//'.DAT'
           end if
        end if
        if(start_file.ne.0)wildcard = .TRUE.
C.....
C---    Check to see if it's a disk or a magtape file.
C
        disk = 0
        if(index(ifile,':').eq.0.or.(ifile(1:2).ne.'MS'.and.
     &  ifile(1:2).ne.'MT'))disk = 1
C.....
C---    If disk = 1 go to the diskfile oper section, otherwise proceed with
C---    a magtape oper and read.
C
        if(disk.ne.1)then
           chan='1'//ifile(1:5)
           call mtfile(input_channe,,raw_header, ifile, wfile,itime,
     &          verify,start_file,stop_file
        else
C.....
C---    Oper and read from the disk file if tape is not the input medium.
C
           if(wildcard.then
              call disk_wildcard(ifile,itime,wfile)
           else
              wfile(1:len2(ifile)) = ifile(1:len2(ifile))
           end if
           call dbmute(5
           call dboper(disk_input_channe,,wfile
           if(ierr.ne.1 then
             type*,'Error on file oper
             disk_input_channe, = 0
             ierr = 1
             go to 12
           end if
C.....
C---    Determine the number of blocks in the input file.
C
           call dbs_re(disk_input_channe,,nblocks
           if(nblocks.eq.0 stop '*** fatal -- input file is empty'
C.....
C---    Read the header from the input file.
C
           call dbread(disk_input_channe,,raw_header,1
           call dbwait(disk_input_channe,
        end if
C.....
C---    Put the byte label into temporary storage for later modification.
C
        do i = 1,2*
           store_blabel(i:i) = blabel(i:i)
        end do
        end if
C.....
C---    Put the screen and write a header message for wildcard files after
C---    the first.
C
        if(itime.gt.1 and.wildcard and.outspec then
           call erase_screen(1,1)
           call set_cursor(2,1)
           call vtmess('re','c',' File Setup '                 'VT100/200 SPECIFIC
        end if
C.....
C---    Allow the user to accept and or change the default output filename.
C
        if(itime.eq.1.and.wildcard)then
           default = 'N'
```

```
            extn = getyn(
     &        ' Default output file extension is TFR - Change it?',
     &        ' All output files will have the specified extension',
     &      default)
            if(extn)then
               call inputs(
     &           ' Enter the new default file extension. ',
     &           default_extension,
     &              'Typically three letters long (leave out the .)')
            end if
            if(wildcard)then
               default = 'N'
               outspec = getyn(
     &           ' Re-specify output parameters for each input file?',
     &           ' Else use the default output filename.',
     &           default)
            end if
         end if
c-----------------------------------------------------------------------
c-- Set the default output filename to be the same as the input filename
c-- but with the extension 'default_extension'
c
         i = index(wfile,'.')
         if(i.eq.0) i = len2(wfile)
         ofile = wfile(max(1,index(wfile,'/')+1):i)//
     &      default_extension(1:len2(default_extension))
         if(itime.eq.1.or.outspec)then
c-----------------------------------------------------------------------
c-- Get the user's choice for the output filename.
c
            call inputs(prompt(3),ofile,help(3))
c-----------------------------------------------------------------------
c-- Determine whether the output file is to be Vax or READRT format.
c
            otype = 'V'
            call inputs(prompt(5),otype,help(5))
            call str$upcase(otype,otype)
         end if
c-----------------------------------------------------------------------
c-- Open the file in the requested format.
c-- (DBOPENC is a disk I/O routine from NRL, orig.nally written for
c--  PDP-11s in macro.)
c
         if(otype.eq.'V')then
            call dbmute(5)
            call dbopenc(disk_channel_out,ofile)
         else
            open(unit=2,file=ofile,status='new',form='unformatted',
     &        access='sequential')
         end if
c-----------------------------------------------------------------------
c---   Work out parameters using data from the input header.
c
         file_type=1                                                    .DAT
         if((ilabel(10).eq.2).and.(ilabel(11).eq.1))file_type=2         .FTR
         if((ilabel(10).eq.2).and.(ilabel(11).ne.1))file_type=3         .PWR
         number_of_channels = ilabel(7)
         bytes_per_block = ilabel(1) * 2
         blocks_per_record = ilabel(2)/ilabel(1)
         samples_per_input_block = 2.*float(ilabel(1))/
     &    float(ilabel(6)*number_of_channels)
         fractional_blocks = float(bytes_per_block)/
     &             float(number_of_channels*ilabel(6))
         fractional_blocks = fractional_blocks - int(
     &             fractional_blocks)
         if(fractional_blocks.lt.0.00001)then
            blocks_per_set = 1
         else
            blocks_per_set = int((1./fractional_blocks)+0.5)
         end if
         frame_fraction = 2.*float(ilabel(1))/(float(ilabel(6))*
     &             float(ilabel(7))*float(ilabel(8)))
         if(file_type.eq.2)frame_time = ilabel(8)
         if(file_type.eq.3)frame_time = ilabel(6)
         block_time = frame_fraction * frame_time
         frame_size = ifix(float(ilabel(8))*float(ilabel(7))*
     &             float(ilabel(6))/float(bytes_per_block))
         if(frame_size .lt.blocks_per_record) then
            if(disk.eq.1.and.frame_size.lt.1)then
               blocks_to_read = 1
            else
               blocks_to_read = blocks_per_record
            end if
         else
            blocks_to_read = frame_size
         end if
c-----------------------------------------------------------------------
c-- Call routine which gets the desired starting time and steps into the
c-- file to that time.  The routine also determines the total number of
c-- blocks to be read from the input file.
c
         call skipper(input_channel,samples_per_input_block,ilabel(1),
```

```
     &   label,blocks_per_record,cldtime,newtime,disk,iblk,
     &           file_type,outspec,itime,total_blocks,block_time,
     &           frame_time,fractional_blocks,number_of_channels)
c.......................................................................
c---   Now allow the user to choose the number of channels to put into
c---   the output file

       call channel_select(blabel,Number_of_channels,outspec,itime,
     &           Channels_to_process,process,hydrophones,Sift_flag,wfile,
     &           wildcard,skip_file)
       if(skip_file.eq.1)go to 102
c.......................................................................
c---   Modify the header to reflect changes.

       ilabel(1) = 256
       if(ilabel(8).eq.1)then
c.......................................................................
c---   Make sure the record size in the header is an integral number of
c---    blocks.

          ilabel(2) = ilabel(2)*channels_to_process/number_of_channels
          if(mod(ilabel(2),256).ne.0)ilabel(2)=256
       end if
       ilabel(7) = channels_to_process
       if(ilabel(3).le.0)ilabel(3) = -1
c.......................................................................
c---   Sort the process vector

       do k =1,(channels_to_process-1)
          do j = (K+1),channels_to_process
             if(process(j).lt.process(k))then
               hold = process(k)
              process(k) = process(j)
              process(j) = hold
            end if
          end do
       end do
       i = 1
       do j = 1 , channels_to_process
          k = (2*process(j)) - 1
          blabel(i) = store_blabel(k)
          blabel(i+1) = store_blabel(k+1)
          i = i + 2
       end do
       do j=((2*channels_to_process) + 1),128
          blabel(j) = 0
       end do
c.......................................................................
c---   Form the selector_mask vector.
c
       k = 1
       jx = 1
       do j = 1,number_of_channels
          if(jx.eq.process(k))then
             selector_mask(j) = 1
             k = k + 1
          else
             selector_mask(j) = 0
          end if
          jx = jx + 1
       end do
c.......................................................................
c---   Write header data into the output file
c
       if(otype.eq.'V')then
          call [illegible]write(disk_channel_out,raw_header,1)
         call [illegible]wait(disk_channel_out)
       else
           write(2)(raw_header(i),i=1,512)
       end if
       [illegible]e_channel_bytes = ilabel(6)*ilabel(8)
       [illegible]utive_channel_bytes * number_of_channels
       [illegible]e[illegible] = blocks_to_read * bytes_per_block
       records_per_read = float(blocks_to_read
     &           [illegible]at blocks_per_record
       [illegible]ds_per_read
c.......................................................................
c---   Determine the number of bytes we need to put into the output file.
c---   Set [illegible] (the size of the output vector) if we have more than
c---   1024 .

       [illegible]es_per_input_block * ilabel(6) *
     & [illegible]process * total_blocks
       jmin = min(10240,jmin)
c.......................................................................
c---   See [illegible] enough input data to write out [illegible] blocks at
c---   a [illegible] do, set default # output blocks to 20, otherwise
c---   set [illegible] number we can fill.

       ob[illegible] ifix((float(jfin) 512.) + 0.[illegible]
       ob[illegible] = min(20,oblocks
       fix[illegible]ead = ifix( float(total_blocks
```

```
     &    float(blocks_to_read)) + 0.5)
          j = 1
          i = 1
          k = 1
          quit_flag = 0
          quit_flag2 = 0
          jrec = 1
          raw_bytes = 0
          call reader(blocks_to_read,bytes_per_block,disk,raw_bytes,
     &       quit_flag,jrec,iblk,nblocks,quit_flag2,input_channel,
     &       disk_input_channel,blocks_per_set)
          call transfer_status(alabel,oldtime,newtime,
     &       channels_to_process,number_of_channels,process,
     &       hydrophones,frames_to_read)
          if(quit_flag2.eq.1)go to 102
          if(quit_flag2.eq.2)go to 43

C.....................................................................
C-- Now throw out any channels we don't want and write the resulting data
C-- to the output disk file.  (watch out for multiplexed or demultiplexed
C-- data.
C
49        if(jrec.gt.frames_to_read)go to 102
          if(selector_mask(k).ne.0)then
             do l=1,consecutive_channel_bytes
                output_data(j) = raw_byte_data(i)
                i = i + 1
                j = J + 1
                if(i.gt.raw_bytes)then
                   if(quit_flag.eq.1)go to 43
                   raw_bytes = 0
                    call reader(blocks_to_read,bytes_per_block,disk,
     &                 raw_bytes,quit_flag,jrec,iblk,nblocks,quit_flag2,
     &                 input_channel,disk_input_channel,blocks_per_set)
                   jrec = jrec + 1
                   nrec = nrec + records_per_read
                   if (mod(nrec,10.).eq.0.0)then
                      call transfer_status(alabel,oldtime,newtime,
     &                 channels_to_process,number_of_channels,process,
     &                 hydrophones,frames_to_read)
                   end if
                   i = 1
                    if(quit_flag2.eq.1)go to 102
                    if(quit_flag2.eq.2)go to 43
                end if
                 if(j.gt.ifin)then
43                oblocks = (j/512)
                  call output(disk_channel_out,otype,oblocks,tblocks)
                   if(quit_flag.eq.1)go to 102
                   j = 1
                 end if
             end do
          else
             i = i + consecutive_channel_bytes
             if (i.gt.raw_bytes)then
50              raw_bytes = 0
                call reader(blocks_to_read,bytes_per_block,disk,raw_bytes,
     &          quit_flag,jrec,iblk,nblocks,quit_flag2,input_channel,
     &          disk_input_channel,blocks_per_set)
                jrec = jrec + 1
                nrec = nrec + records_per_read
                if (mod(nrec,10.).eq.0.0)then
                   call transfer_status(alabel,oldtime,newtime,
     &                channels_to_process,number_of_channels,process,
     &                hydrophones,frames_to_read)
                end if
                 if(quit_flag2.eq.1)go to 102
                 if(quit_flag2.eq.2)go to 43
                i = i - raw_bytes
                if(i.gt.raw_bytes)go to 50
             end if
          end if
             k = k + 1
             if(k.gt.number_of_channels)k = 1
C.....................................................................
C-- Update program status on terminal
C
          go to 49
C.....................................................................
C-- If wildcard process next file - otherwise let user choose to
C-- Process another or exit.  Also close the input disk file.
C
102       itime = itime + 1
          if(disk.eq.1)then
             call dbclose(disk_input_channel)
          end if
          if(otype.eq.'V')then
C.....................................................................
C-- Close output file, but first calculate the number of records in the
C-- file and rewrite the header block
C
             ilabel(3) = tblocks * ilabel (1) / ilabel (2)
```

```
            call dbwrite(disk_channel_out,raw_header,1,ierr,0)
            call dbwait(disk_channel_out)
            call dbclose(disk_channel_out)
         else
            close(unit=2)
         end if
         if(wildcard)then
            go to 18
         else
            default = 'N'
            another = getyn(
     &      '  Do you want to process another file?',
     &      '  Default is to exit the program.',
     &      default)
            if ( another) go to 19
         end if
C.....................................................................
C-- Dismount tape if one was used.
C
         if(disk.ne.1)then
         write(5,*)'  Dismounting Tape'
            mask = dmt$m_nounload
            status = sys$dismou(chan,%val(mask))
            if(.not.status)call lib$stop(%val(status))
         end if
         stop
         end


C
C-----------------------------------------------------------------------
C
         subroutine check(iosb,ipos)
C
C      This routine checks the status word of the iosb (io status buffer)
C   buffer returned by system calls and aborts the program if problems occur.
C
         integer*2 iosb(4)

         if(iosb(1).ne. 1) then
           type*,' error in system call at position',ipos
           do i = 1,4
             type100,i,iosb(i)
100          format(5x,'text_iosb(',i2,') = ',i8)
           enddo
           stop
         endif
         return
         end
```

B2 - Listing of Data Output Routine

```
C**********************************************************************
C                        Subroutine NAME: OUTPUT                       *
C**********************************************************************

C
C                          Written by:
C                        Joseph B. Farrell
C                              DREA
C                            15 Feb. 1986
C
C                   Latest revision:  7 Aug. 1986

         subroutine output(output_channel,filetype,oblocks,tblocks)
C
C        This routine writes sorted output data into a disk file using
C   the DBMT routines from NRL.


         byte              output_data(10240)

         character         filetype

         integer*4         oblocks
         integer*4         output_channel
         integer*4         tblocks

         common /odata/output_data
         tblocks = tblocks + oblocks
C.....................................................................
C-- Write the sorted output data to the disk file.
C
         if(filetype.eq.'V'.or.filetype.eq.'v')then
            call dbwrite(output_channel,output_data,oblocks)
            call dbwait(output_channel)
         else
            write(2)(output_data(lx),lx=1,(512*oblocks))
         end if
```

```
      return
      end
```

B3 - Listing of Mag-Tape File Open Routine "MTFILE"

```
C*********************************************************************
C                      Subroutine NAME: MTFILE                       *
C*********************************************************************

C
C                        Written by:
C                  D. Vance Crowe & Joseph B. Farrell
C                             DREA
C                          12 Mar. 1986
C
C                   Latest revision:  15 Mar. 1986

      SUBROUTINE MTFILE(input_channel, header, ifile, wfile, itime,
     &           verify,start_file,stop_file )
C
C  This subroutine opens a DREA data file from mag tape.
C
C.....................................................................
C---   Parameter and internal variable declarations.
C
      IMPLICIT INTEGER*4 (a-z)

      PARAMETER MSG=5

      BYTE              IC0                     !Null byte = Null character
      BYTE              buffer(512)             !Raw header data
      BYTE              header(512)             !Raw header data

      CHARACTER*80      label                   !File name labels.
      CHARACTER         chan*6                  !Used in mag tape channel
      CHARACTER         default*1               !User input default
      CHARACTER         help(msg)*80            !User help prompts
      CHARACTER         ifile*64                !Input filename
      CHARACTER         icnull                  !Null byte = Null character
      CHARACTER         prompt(msg)*80          !User prompts
      CHARACTER         tape_mark(10)           !Checks tape marks
      CHARACTER         tape_name(10)           !Header filename
      CHARACTER*12      VOL1_HDR1               !VOL1, HDR1 and EOF1 labels
      CHARACTER         wfile*64                !working file name

      INTEGER*2         file_count
      INTEGER*2         input_channel           !Input channel for tape
      INTEGER*2         text_Iosb(4)            !Status buffer
      INTEGER*2         iskip                   !skip count
      INTEGER*2         start_file
      INTEGER*2         stop_file

      INTEGER           what_to_do              !option flag for file search

      LOGICAL           end_of_tape             !End of tape indicator
      LOGICAL           correct_file            !File flag
      LOGICAL           getyn                   !User response function
      LOGICAL           skip                    !Skip file flag
      LOGICAL           verify

      equivalence ( tape_name(1),buffer(5) )
      equivalence ( label,buffer(1) )
      equivalence ( ic0,icnull )

      common /io_statistics/jrec,nrec

      external   io$_skipfile,io$_readvblk,io$_skiprecord,io$_rewind
C
      Data VOL1_HDR1/'VOL1HDR1EOF1'/, IC0/0/, what_to_do/-1/
C.....................................................................
C---   Data for user interface.
C
     data (prompt(i),i=1,msg)/
     &'  Enter name for the input datafile.',
     &'  Want to pass over this file?(-1=y/0=only/1=no/2=rewind)',
     &'  Enter a name for the Disk output file.',
     &'  How many data frames do you want to read?',
     &'  Should the output file be Vax (V) or READRT (R) format?'/

     data (help(i),i=1,msg)/
     &'  File name: UNIT or STRUCTURE:NAME.EXTENSION',
     &'  -1=pass file&look at next/0=correct/1=use file/2=rewind.',
     &'  Default name will be the same as the input filename.',
     &'  Frames to read from the input file.',
     &'  (V) format compatible with DISPVAX, SASPEC, etc.'/

C     data ifile/'msa0:q38334.dat'/
C
          chan='_'//ifile(1:5)
```

```
        if(itime.eq.1)then
           file_count = 0
           call gettap(chan)
C.....................................................................
C---   Assign a channel to the mag tape
C
           status=sys$assign(chan,input_channel,,)
           if(.not.status)call lib$stop(%val(status))
        end if
        if(file_count.gt.stop_file)stop 'All files processed.'
C.....................................................................
C---   Read the RT-11 header (incl. filename) from tape into 'buffer'
C
110        ierror = 0
           status=sys$qiow(,%val(input_channel),io$_readvblk,text_iosb,,
     1       ,%ref(buffer),%val(512),,,,)
           if(.not.status)call lib$stop(%val(status))
C
                if(.not.status.or.text_iosb(2).eq.0)then  !there was an error
                  ierror = 1
                     if(text_iosb(2).ne.0)then  !there was a tape read error
                     else                       !or there was an EOF
                     end if
                end if

        If(ierror.Eq.1) Go to 110  !Try again
C
        itype = INDEX( VOL1_HDR1, label(1:3))
        If(itype.Eq.0)then
           If ( itime.eq.1 )then
             iskip = -2            !Probably a data record - go back to file start
           else
             iskip = +2            !Probably a data record - go ahead to file st
           end if
        end if
        If(itype.Eq.1 ) Go to 110  !VOL label, read next header
        If(itype.Eq.5 ) Go to 150  !HDR label, check names etc
        If(itype.Eq.9 ) iskip = +1 !EOF label, skip forward 1 EOF
C   !Skipping file marks
120             status=sys$qiow(,%val(input_channel),io$_skipfile,
     1           text_iosb,,,%val(iskip),,,,,)
                if(.not.status)call lib$stop(%val(status))
                call check(text_iosb,-3)
              Go to 110
C.....................................................................
C---   Check to see if we've reached the end of this tape (Filename all 0's)
C
150        end_of_tape = .true.
           do I = 1,10
              if (tape_name(i) .ne. icnull) then
                 end_of_tape = .false.
              endif
           end do
           if (end_of_tape) then
             type*,'End of tape encountered,  Rewinding.'
C
C   Rewind tape!...
160        status=sys$qiow(,%val(input_channel),io$_rewind,text_iosb,,,,,,,,)
           if(.not.status)call lib$stop(%val(status))
           what_to_do = -1
           Go to 110               !Read the volume labels
        end if
C.....................................................................
C---   If not at end of the tape, check to see if we're picking a
C---   numbered file.  If we are, position the tape to that file.
C
        if(start_file.gt.1.and.itime.eq.1)then
           iskip = 3 * (start_file - 1)
           file_count = 1
           type*,'  Skipping to next file.'
           Go to 120
        end if
        if(start_file.eq.1.and.itime.eq.1)file_count = 1
C.....................................................................
C---   See if we're reading the correct file...  (wildcards allowed)
C
               correct_file = .true.
C.....................................................................
C---   If we're doing numbered files, we don't have to check the filename.
C
               if(start_file.ne.0)go to 2
               i = index(ifile,':') + 1
               period_flag = 0
               name_length = len2 ifile
               do i = 1,10
                  i2=ichar(ifile(i:
C.....................................................................
C---   Check for wildcard '*' character in input filename.
C
                  if(i2.ne.42)the
C.....................................................................
C---   Compare input and tape filenames, if after if input not '*'.
```

```
C
                    il=ichar(tape_name(i))
                    if (il.ne.i2) then
                       correct_file = .false.
                        go to 21
                     end if
                  else
C.....................................................................
C---   If we got a wildcard character, look for '.' if period_flag = 0.
C
                    if(period_flag.eq.0)then
                       do while (ichar(tape_name(i)).ne.46.and.
     &                    i.lt.10)
                          i = i + 1
                       end do
                       i = i - 1
                       period_flag = 1
                     else
C.....................................................................
C---   If we've already passed the '.' then filename is assumed correct.
C
                       go to 20
                     end if
                   end if
                  j = j + 1
                   if(j.gt.name_length)go to 20
                end do
C   If this was the correct file, indicate success and proceed.
              if ( correct_file) then
20               do k = 1,index(ifile,':')
                     wfile(k:k) = ifile(k:k)
                 end do
                 do k = 1,10
                    j=k+ index(ifile,':')
                    wfile(j:j) = tape_name(k)
                 end do
               type*,' file ',wfile(6:15),' found'
                  goto 220
            end if
C.....................................................................
C---   If it's not the right file, skip to the next one (giving the user
C---   the option of processing the file which has been found).
C
21       continue
         if(verify)then
                  type*,'  Found file: ',tape_name
                   If(what_to_do.Eq. -1) Then !Ask the question the first time
                   call inputI(prompt(2),what_to_do,help(2))
                     Endif
         else
                  what_to_do = 0
         end if
                  If(what_to_do .Eq. 0) Go to 180     !Find the correct file
                  If(what_to_do .Eq. 1) Go to 190     !Use this file as correct one
                  If(what_to_do .Eq. 2) Go to 160     !Rewind tape.
                  If(what_to_do .Ne.-1) Then
                           Type *,' Assume you want to skip to next file.'
                           endif
180                        iskip = +3
            type*,'  Skipping to next file.'
                           Go to 120                  !Skip forward 3 file marks.
C
C  The user wants this file that he/she found.
190               do k = 1,index(ifile,':')
                      wfile(k:k) = ifile(k:k)
                  end do
                  do k = 1,10
                    j=k+ index(ifile,':')
                    wfile(j:j) = tape_name(k)
                  end do
                  go to 220
C.....................................................................
C---   Skip over file mark to the beginning of the drea header data
C
220            status=sys$qiow(,%val(input_channel),io$_skiprecord,
          1       text_iosb,,,%val(1),,,,,)
                if(.not.status)call lib$stop(%val(status))
C.....................................................................
C---   We're finally in position, so read in the DREA header
C
                status=sys$qiow(,%val(input_channel),io$_readvblk,text_iosb,,
          1     ,%ref(buffer),%val(512),,,,)
                if(.not.status)call lib$stop(%val(status))
D                  Type *,' finished reading drea header'
                call check(text_iosb,-5)
C
C.....................................................................
C---   At this point, we've read in the header from tape.
C
         file_count = file_count + 1
         DO 310 ii=1,512
310      header(ii)=buffer(ii)
```

```
C
      RETURN
      END
```

B4 - Routine to Allocate and Mount Mag-Tape

```
C*****************************************************************
C                        Subroutine NAME: GETTAP                  *
C*****************************************************************
C
C                          Written by:
C                        Joseph B. Farrell
C                             DREA
C                           15 Mar. 1986
C
C                 Latest revision:  7 Aug. 1986

      subroutine gettap(chan)

      structure /itmlst/
       union
        map
         INTEGER*2       buflen
         INTEGER*2       code
         INTEGER*4       bufadr
         INTEGER*4       endlst
        end map
        map
         INTEGER*4       end_list
        end map
       end union
      end structure

      record /itmlst/ mnt_list(3)
      include '($mntdef)'
      include '($dmtdef)'
      include '($ssdef)'

      CHARACTER         chan*6                    !Used in mag tape channel
      INTEGER*4         mask /0/
      INTEGER*4         status
      INTEGER*4         sys$mount
      INTEGER*4         sys$alloc

C.....................................................................
C---   Allocate and mount the mag tape.
C
      status = sys$alloc(chan,,,,)
C.....................................................................
C---   If tape is already mounted skip mounting section.
C
      if (status.eq.ss$_devmount)then
         type*,'Tape is already mounted - assuming you did it.'
         go to 10
      end if
      if (status.eq.ss$_devalloc)then
         write(5,*)' Error - device probably allocated to another user'
         stop
      end if
      if (status.eq.ss$_devalralloc)then
         write(5,*)'Device already allocated to you.'
      end if
      mask = 0
      mask = mnt$m_foreign .or. mnt$m_message .or. mnt$m_noassist

      mnt_list(1).buflen = 6
      mnt_list(1).code = mnt$_devnam
      mnt_list(1).bufadr = %loc(chan)
      mnt_list(1).endlst = 0
      mnt_list(2).buflen = 4
      mnt_list(2).code = mnt$_flags
      mnt_list(2).bufadr = %loc(mask)
      mnt_list(2).endlst = 0
      mnt_list(3).end_list = 0

      status = sys$mount(mnt_list
      if(.not.status)call lib$stop(%val(status))
10    continue
      return
      end
```

B5 - Channel Selection Subroutine

```
C*****************************************************************
C                        SUBROUTINE NAME: Channe_Se.e*            *
```

```
C*****************************************************************
C
C                          Written by:
C                       Joseph B. Farrell
C                            DREA
C                          27 Feb. 1986
C
C                   Latest revision:  8 Jul. 1986
C
C       This subroutine chooses which of the channels in an RT-11 or VAX
C    format data file are to be processed.  It uses the GETVEC subroutine
C    written by Doug  Peters (ASP Summer Student) to read a vector from the
C    user's terminal.
C
C   SUBROUTINE CALLS:
C                 1)  INPUTI      = Routine which reads an integer from the
C                                   terminal.
C                 2)  GETVEC      = Routine which reads a vector from the
C                                   terminal.
C                 3)  WAIT = Waits for a specified amount of time.
C                 4)  Erase_screen = Clears the terminal screen.
C                 5)  Set_cursor   = Moves the cursor to a specified location
C
C*****************************************************************
C                            MAIN CODE
C*****************************************************************
C
      subroutine channel_select(blabel,number_of_channels,outspec,itime,
     &          channels_to_process,process,hydrophones,sift_flag,wfile,
     &          wildcard,skip)
C.....................................................................
C---  Parameter and internal variable declarations.
C
      implicit integer*4 (a-z)

      parameter msg=1

      byte              blabel(128)

      character         default*1
      character         help(msg)*70
      character         wfile*64
      character         prompt(msg)*70
      character         response*3
      character         string*12

      integer*2         channels(128)
      integer*2         original_channels(128)
      integer*2         hydrophones(128)
      integer*2         new_HP
      integer*2         number_of_channels
      integer*2         number_of_hydrophones
      integer*2         phones(128)
      integer*2         original_phones(128)
      integer*2         process(128)
      integer*2         sift_flag

      integer*4         channels_to_process
      integer*4         original_channels_to_process
      integer*4         check_channel
      integer*4         finish
      integer*4         iflag
      integer*4         ip

      LOGICAL           all_flag
      LOGICAL           ok(128)
      LOGICAL           outspec
      LOGICAL           preserve_phones
      LOGICAL           proceed
      LOGICAL           wildcard

      real*4            val(128)
C.....................................................................
C---  Data for user interface.
C
     data (prompt(i),i=1,msg)/
     &' How many channels do you want to process? (-1 for all)'/

     data (help(i),i=1,MSG)/
     &' Enter the number of channels to process.'/
C.....................................................................
C---  Variable initializations
C
      number_of_hydrophones = 0
      new_HP = 0
      if(itime.eq.1.or.outspec)all_flag = .FALSE.
C.....................................................................
C---  Determine which channels contain acoustic data. Collect the
C---  hydrophone numbers and corresponding channels.
C
      do  i=1,2*number_of_channels,2
          j = i + 1
```

```
            k = j/2
            if (blabel(i) .ge. 0)then
               number_of_hydrophones = number_of_hydrophones + 1
               channels(number_of_hydrophones) = k
               phones(number_of_hydrophones) = blabel(i)
            end if
         end do
C.....................................................................
C---   Must be at least one hydrophone in the file.
C
         if (number_of_hydrophones .lt. 1)
     &    stop 'Channel_select - no acoustic data found in file.'

C.....................................................................
C---   Allow channel numbers or H/P numbers to be preserved between
C---   files.
C
         if(.not.outspec.and.itime.eq.1.and.wildcard)then
            default = 'Y'
            preserve_phones = getyn(
     &      '  Select the same H/P (Y) or channels (N) from each file?',
     &      '    Default will select the same H/P from each input file.',
     &     default)
         end if
C.....................................................................
C---   If itime > 1 compare hydrophones with those from the original file
C---   and write out a warning if the chosen ones differ.
C
         if(.not.outspec.and.itime.gt.1)then
            missing_flag = 0
            l = 1
C.....................................................................
C---   Look through the available phones to see if the desired ones are
C---   there.
C
            do j = 1, original_channels_to_process
               do k = 1,number_of_hydrophones
                  if (phones(k).eq.original_phones(j))then
C.....................................................................
C---   Use the process vector to point at the location of the desired
C---   H/P if we're keying on H/P.
C
                     if(preserve_phones)then
                        process(l) = k
                        l = l + 1
                     end if
                     go to 7
                  end if
               end do
C.....................................................................
C---   Set a flag to indicate H/P missing if we can't find one.
C
               missing_flag = 1
               continue
            end do
C.....................................................................
C---   If there are H/P missing write a warning and let the user decide
C---   what to do.
C
            if(missing_flag.eq.1)then
               call erase_screen(1,1)
               call set_cursor(2,1)
               call vtmess('re','c',' Channel Setup ')
               type 1000,wfile
1000           format('   Working on file: ',A20)
               type*,'  The h/p available differ from the originals.'
               type *,'  OLD: ',(original_phones(jj),jj=1,
     &         original_channels_to_process)
                 type *,'  NEW: ',(phones(jj),jj=1,
     &           number_of_hydrophones)
               if(.not.preserve_phones)then
                  type*,'  You are keying on channels rather than phones ',
     &            'so I am proceeding.'
                  call wait('0 ::3',5)
               end if
C.....................................................................
C---   If we're preserving a set of H/P betwen files,
C---   Option to proceed with the found subset of H/P or to skip file.
C
               if(preserve_phones)then
                  proceed = .false.
                  if(l.gt.1)then
                     default = 'Y'
                     proceed = getyn(
     &               'Proceed using the subset of requested phones found?',
     &               'Re-specify H/P or skip this file if reply is N.',
     &               default)
                  end if
C.....................................................................
C---   If the user chose not to proceed with the subset found, give
C---   the option on re-specifying the H/P or of skipping the file.
C
```

```
                  if(.not.proceed)then
                     default = 'Y'
                     proceed = getyn(
     &               'Skip to the next file in the input set?',
     &               'Default is to request a new set of H/P to process.',
     &               default)
                     if(proceed)then
                         skip = 1
                         return
                     else
                         new_HP = 1
                        go to 8
                     end if
                  end if
               end if
C.....................................................................
C---   If we're using the subset, continue processing.
C
            end if
C.....................................................................
C---   If we're keying on channels rather than H/P make sure we have
C---   enough channels.
C
          if(.not.preserve_phones)then
C.....................................................................
C---   Only do it if we're not processing all channels in the file.
C
            if(.not.all_flag)then
                  k = 1
                  do j = 1,original_channels_to_process
C.....................................................................
C---   Check to see if a requested channel is > than the # we have.
C
               if(original_channels(j).gt.number_of_hydrophones)then
                  type*,' You asked for a channel not found in the',
     &             ' input file.'
C.....................................................................
C---   If channel out of range, user can skip file or proceed.
C
                     default = 'Y'
                     proceed = getyn(
     &               'Skip to the next file in the input set?',
     &               'Otherwise use this file with reduced # of chans.',
     &               default)
                     if(proceed)then
                        skip = 1
                        return
                      end if
                  else
                      type*,' Proceding with reduced # of channels.'
                     process(k) = original_channels(j)
                     k = k + 1
                  end if
               end do
            channels_to_process = k - 1
            end if
          end if
        end if
        if(all_flag)then
           channels_to_process = number_of_channels
           go to 5
        end if
C.....................................................................
C-- Clear the screen and write a header message.
C
        if(itime.eq.1.or.outspec)then
8          call erase_screen(1,1)
           call set_cursor(2,1)
           call vtmess('re','c',' Channel Setup ')
C.....................................................................
C---   Show the acoustic channels available and let the user choose some.
C
           type 1, number_of_channels
1          format(' ',t3,' There are',i3,' channels in the input file.')

           type 2, (channels(i),i=1,number_of_hydrophones)
2          format(' ',t3,' The following are acoustic channels: '/
     &          ' ',t3,26(i2:',')/' ',t3,26(i2:',')/' ',t3,26(i2:','))

           type 3, (phones(i),i=1,number_of_hydrophones)
3          format(' ',t3,' The corresponding H/P numbers are: '/
     &          ' ',t3,26(i2:',')/' ',t3,26(i2:',')/' ',t3,26(i2:','))

           channels_to_process = -1
4          call inputi(prompt(1),channels_to_process,help(1))
        end if
C.....................................................................
C---   Set flag indicating all channels to be processed for subsequent
C---   files if necessary.
C
        if(itime.eq.1.or.new_HP.eq.1.or.outspec)then
           if(channels_to_process.eq.-1)then
```

```
                all_flag = .TRUE.
                channels_to_process = number_of_channels
              end if
              original_channels_to_process = channels_to_process
            end if
C.....................................................................
C---   User must choose hydrophones >= 1 and <= number available
C
6           if (channels_to_process .lt. 1 .or.
     &          channels_to_process .gt. number_of_channels) then
              type 3500
3500          format(/' channel_select  - illegal number of channels to',
     &               ' process chosen.'/)
              call wait('0 ::2',5)
              call erase_screen(5,1)
              go to 4
C.....................................................................
C---   Load all H/P, channels and gains if all are to be analyzed.
C
            else if (channels_to_process .eq. number_of_channels) then
5             do 400 i=1,number_of_channels
                process(i) = i
                hydrophones(i) = phones(i)
                if(itime.eq.1)then
                  original_phones(i) = phones(i)
                  original_channels(i) = process(i)
                end if
400           continue
              return
            end if
C.....................................................................
C---   Identify individual h/P if a subset of the total was chosen.
C
            if(itime.eq.1.or.new_HP.eq.1.or.outspec)then
              type 4000
4000          format(/' ',t3,' Enter the channels you wish to study : ',$)
              call getvec(val,ok,channels_to_process)
              if (ok(1)) then
                do j=1,channels_to_process
                  process(j)=int(val(j))
                end do
C.....................................................................
C---   Set flag to indicate that a channel(s) must be removed from data.
C
                sift_flag = 1
              else
                type '(/'' channel_select - sorry, no default.'')'
                call wait('0 ::2',5)
                call erase_screen(5,1)
                go to 4
              end if
            end if
              do j=1,channels_to_process
                hydrophones(j) = phones(process(j))
                if(itime.eq.1)then
                  original_phones(j) = phones(process(j))
                  original_channels(j) = process(j)
                end if
              end do
            return
            end
```

B6 - Routine For Choosing Data Segment To Be Transferred

```
C**********************************************************************
C                       SUBROUTINE NAME:  SKIPPER                     *
C**********************************************************************
C
C                            Written by:
C                          Joseph B. Farrell
C                                DREA
C                             14 Feb. 1986
C
C                     Latest revision:  23 May. 1986
C
C        This program uses the header to determine the starting time of an
C   input file, asks the user to enter a desired start time, and then steps
C   into the file the desired amount.
C
C**********************************************************************
C                                MAIN CODE
C**********************************************************************
C
        subroutine skipper(input_chan,points_per_block,sampling_frequency,
        1 alabel,blocks_per_record,oldtime,newtime,disk,disk_start_block,
        1 file_type,outspec,itime,total_blocks,block_time,frame_time,
        1 fractional_blocks,number_of_channels)
C.....................................................................
C---   Parameter and internal variable declarations.
```

```
C
      IMPLICIT INTEGER*4 (a-z)

      PARAMETER msg=2

      CHARACTER         alabel*320
      CHARACTER         default*1
      CHARACTER         help(msg)*70
      CHARACTER         newtime*8
      CHARACTER         oldtime*8
      CHARACTER         transtime*8
      CHARACTER         prompt(msg)*70

      INTEGER           blocks_per_record
      INTEGER           blocks_to_skip
      INTEGER*4         disk
      INTEGER*4         disk_start_block
      INTEGER*2         file_type
      INTEGER*2         input_chan
      INTEGER*4         itime
      INTEGER*2         new_time(3)
      INTEGER*2         number_of_channels
      INTEGER*2         redo
      INTEGER*2         text_iosb(4)
      INTEGER*4         total_blocks
      INTEGER*2         transfer_time(3)

      LOGICAL           getyn
      LOGICAL           newtim
      LOGICAL           outspec
      LOGICAL           status

      REAL              begin_time
      REAL              block_time
      REAL              fractional_blocks
      REAL              frame_time
      REAL              old_time(3)
      REAL              points_per_block
      REAL              blocks_per_hour
      REAL              real_skip
      REAL              sampling_frequency
      REAL              tape_time
      REAL              test
      REAL              time_difference

     EXTERNAL   io$_skipfile,io$_readvblk,io$_skiprecord,io$_rewind
C.....................................................................
C---   Data for user interface.
C
     data (prompt(i),i=1,msg)/
    &'  Do you want to begin processing at some other time.',
    &'  Enter the time at which you wish to start.'/

     data (help(i),i=1,msg)/
    &'  Default is to start at the time shown.',
    &'  Format is HH:MM:SS.'/
C.....................................................................
C---   Variable initializations.
C
      newtim = .false.
      oldtime(1:) = alabel(12:19)
      newtime(1:) = alabel(12:19)
      if(file_type.eq.1)then
         blocks_per_hour = 3600.*sampling_frequency/
     1    (points_per_block)
         block_time = 1./blocks_per_hour
      end if
C.....................................................................
C---   Print the file start time on the user's terminal.
C
      newtim = .FALSE.
      if(itime.eq.1.or.outspec)then
         call erase_screen(1,1)
         call set_cursor(2,1)
         call vtmess('re','c',' This file starts at '//alabel(12:19),' ')
C.....................................................................
C---   Let the user determine a start time for data analysis.
C---   (If NEWTIM is returned as "FALSE" analysis starts at the beginning
C---   of the file.)
C
         default = 'N'
         newtim = getyn(prompt(1),help(1),default)
      end if
      if( newtim )then
C.....................................................................
C---   Decode the file start time into the vector Old_time.
C
          decode(2,1000,alabel(12:13))old_time(1)
          decode(2,1000,alabel(15:16))old_time(2)
          decode(2,1000,alabel(18:19))old_time(3)
1000      format(f2.0)
C.....................................................................
```

```
C---   If we're changing start times, read in the new time and "DECODE" it
C---   into the array New_time.
C
1          call inputs(prompt(2),newtime,help(2))
           if(newtime.eq.oldtime)go to 1
           decode(2,1001,newtime(1:2))new_time(1)
           decode(2,1001,newtime(4:5))new_time(2)
           decode(2,1001,newtime(7:8))new_time(3)
1001       format(I2)
C.....................................................................
C---   Tape_time is set to the input file start time (in decimal hours).
C---   Begin_time is set to the processing start time.
C
           tape_time = old_time(1) + (old_time(2) / 60.0) +
     1      (old_time(3) / 3600.0)
           begin_time = float(new_time(1))+(float(new_time(2))/ 60.0)+
     1      (float(new_time(3)) / 3600.0)
           time_difference = begin_time - tape_time
C.....................................................................
C-- Make sure we start on a block which begins with the first channel
C-- if we're doing a .DAT file.
C
        nblocks = 1
        if (fractional_blocks.gt.0.0001)then
           do nblocks = 2,number_of_channels
              test = float(nblocks)*fractional_blocks
              if( (abs(test) - abs(int(test))).lt.0.0001)then
                 go to 22
              end if
           end do
        end if
C.....................................................................
C-- Check to make sure we aren't moving into the middle of an FFT or
C-- a spectrum if the file is .FTR or .PWR
C
        if(file_type.eq.2)call timer(time_difference,frame_time,redo)
        if(file_type.eq.3)call timer(time_difference,frame_time,redo)
        if(redo.eq.1)go to 21
C.....................................................................
C---   Determine how many records to skip before beginning processing.
C---   (file_type=1 indicates .DAT, =2 indicates .FTR, and =3 .PWR)
C
22      if(file_type.eq.1)then
           blocks_to_skip = int(blocks_per_hour * time_difference)
C.....................................................................
C-- Make sure we start on a block which begins with the first channel
C-- if we're doing a .DAT file.
C
24            if(amod(float(blocks_to_skip),float(nblocks)).eq.
     &       0.)then
              go to 23
           else
              blocks_to_skip = blocks_to_skip - 1
              go to 24
           end if
23         real_skip=tape_time + (float(blocks_to_skip)/blocks_per_hour)
        else
           blocks_to_skip = (int(time_difference / block_time))
           real_skip=tape_time + (float(blocks_to_skip) * block_time)
        end if
C.....................................................................
C---   Set the start time to the time we're actually going to skip
C---   into the file (We may not be able to skip exactly to the requested
C---   time because of the finite record lenght in the input file).
C
        new_time(1) = int(real_skip)
        real_skip = (real_skip-new_time(1))*60
        new_time(2) = int(real_skip)
        real_skip = (real_skip-new_time(2))*60
        new_time(3) = nint(real_skip)
C.....................................................................
C---   "ENCODE" the actual start time into the character sting NEWTIME
C---   and display it on the user's terminal.
C
           encode(2,1001,newtime(1:2)) new_time(1)
           encode(2,1001,newtime(4:5)) new_time(2)
           encode(2,1001,newtime(7:8)) new_time(3)
           type*,'  Actual start time will be ',newtime
           alabel(12:19)=newtime(1:8)
C.....................................................................
C---   Skip the requested number of records
C
        if(disk.ne.1)then
C.....................................................................
C---   Skip records on tape if that is the media being used.
C
           if (blocks_to_skip .ne. 0) then
           type*,blocks_to_skip,' physical blocks will be skipped.'
             status=sys$qiow(,%val(input_chan),io$_skiprecord,text_iosb,,
     1        ,%val(blocks_to_skip),,,,,)
             if(.not.status)call lib$stop(%val(status))
             call check(text_iosb,-11)
```

```
            endif
         else
C.....................................................................
C---   Set the start block for reads from disk.
C
            disk_start_block = blocks_to_skip + 1
            type*,'  Disk start block will be: ',disk_start_block
         end if
         call wait('0 ::2',5)
         else
C.....................................................................
C--   Control jumps here if we're going to start at the beginning of the
C   file.
C
1           disk_start_block = 1
         end if
C.....................................................................
C---   Now give the user the option of specifying the number of records
C---   to be read from the input file or specifiying a time interval.
C
         if(itime.eq.1.or.outspec)then
         total_blocks = -1
         call inputi(
     &   '  Enter n to X-fer n blocks,-1 for all,-2 to specify time',
     &   total_blocks,
     &   '  -2 will let you enter a time interval for the transfer.')
         if(total_blocks.eq.-1)then
            total_blocks = 100000
         else
            if(total_blocks.eq.-2)then
               transtime = '00:01:00'
20             call inputs(
     &         '  Enter the length of time of the transfer.',
     &         transtime,
     &         '  Format is HH:MM:SS.')
              decode(2,1001,transtime(1:2))transfer_time(1)
              decode(2,1001,transtime(4:5))transfer_time(2)
              decode(2,1001,transtime(7:8))transfer_time(3)
              time_difference = transfer_time(1) + (transfer_time(2) / 60.0) +
     1     (transfer_time(3) / 3600.0)
              if(time_difference.eq.0)then
                 type*,'  You specified a zero-length transfer - try again.'
                go to 20
              end if
C.....................................................................
C-- Check to make sure we're transferring at least a full FFT or spectrum
C
         if(file_type.eq.2)call timer(time_difference,frame_time,redo)
         if(file_type.eq.3)call timer(time_difference,frame_time,redo)
         if(redo.eq.1)go to 20
               total_blocks = time_difference / block_time
               if(total_blocks.eq.0)then
                  type*,'  WARNING - you are trying to transfer 0 blocks!',
     &               '  Try again.'
                  type*,'  Transfer time must be at least',
     &               frame_time*3600.
                  go to 20
               end if
            else
               total_blocks = total_blocks * blocks_per_record
            end if
         end if
         end if
         return
         end

         subroutine timer(time_difference,unit_time,redo)

         IMPLICIT INTEGER*4 (a-z)

         INTEGER*2          choose
         INTEGER*2          redo

         REAL               check
         REAL               seconds
         REAL               time_difference
         REAL               unit_time

         redo = 0
         check = amod(time_difference,unit_time)
         seconds = unit_time * 3600.
         if(check.ne.0.)then
            type*,'  You are not using an integral number of records.'
            choose = 1
            call inputi(
     &   '  Type 1 to take closest record start, 2 to re-specify time',
     &      choose,
     &   '  Closest may be earlier or later than the chosen time.')
            if(choose.eq.1)then
               time_difference = anint(time_difference / unit_time) *
     &         unit_time
            else
```

```
            type*,'  Time difference must be a multiple of',seconds,
     &      ' seconds.'
            redo = 1
         end if
      end if
      return
      end
```

B7 - Routine to Decode Switches in Filename

```
C*****************************************************************
C                      SUBROUTINE NAME: SWITCHES                 *
C*****************************************************************

C
C                          Written by:
C                        Joseph B. Farrell
C                             DREA
C                           3 Jul. 1986
C
C                   Latest revision:  3 Jul. 1986
C
C       This subroutine picks switches from a user input filename.

        subroutine switches ( filename,verify,start_file,stop_file)

        character         filename*64

        INTEGER*2         number
        INTEGER*2         start_file
        INTEGER*2         stop_file
        INTEGER*2         start
        INTEGER*2         stop

        logical           verify
C.....................................................................
C---    Initialize variables
C
        verify = .FALSE.
C.....................................................................
C---    Check to see if verify flag is present.
C
        start = index( filename , '/V')
        if( start .ne. 0 ) verify = .TRUE.
C.....................................................................
C---    Look for a START flag , and if present decode the starting file
C---    number.
C
        start = index ( filename , '/START=' )
        if ( start .ne. 0 ) then
           start = start + 7
           stop = index( filename(start:) , '/' )
           if ( stop .eq. 0 ) then
              stop = len2(filename)
              number = stop - (start-1)
           else
              stop = start + stop - 2
              number = stop - (start-1)
           end if
           decode (number,100,filename(start:stop)) start_file
100        format( I2 )
C.....................................................................
C---    Look for a STOP flag , and if present decode the stopping file
C---    number.
C
           start = index ( filename , '/STOP=' )
           if ( start .ne. 0 ) then
              start = start + 6
              stop = index( filename(start:) , ' ' )
              if ( stop .eq. 0 ) then
                 stop = len2(filename)
                 number = stop - (start - 1)
              else
                 stop = start + stop - 2
                  number = stop - (start - 1)
              end if
              decode (number,100,filename(start:stop)) stop_file
           else
              stop_file = start_file
           end if
        end if
C.....................................................................
C---    Look for switch mark and remove all switches from the filename.
C
        start = index ( filename , '/')
        if ( start .ne. 0 ) then
           filename(start:len2(filename)) = ' '
        end if
        return
```

```
      end
```

B8 - Routine to Determine Wildcard File List

```
C*****************************************************************
C                     Subroutine NAME: Disk_Wildcard             *
C*****************************************************************
C
C                        Written by:
C                     Joseph B. Farrell
C                          DREA
C                        10 Jul. 1986
C
C                 Latest revision:  7 Aug. 1986

      subroutine disk_wildcard(ifile,itime,wfile)

      CHARACTER         string*80
      CHARACTER         ifile*64
      CHARACTER         wfile*64

      INTEGER*4         dev_flag
      INTEGER*4         itime
C.....................................................................
C---  Do a directory using the input filspecs of ifile (first time only).
C
      if(itime.eq.1)then
         string='dir/siz/co:1/ou:transfer.tmp '//ifile
         istatus=lib$spawn(%descr(string(1:len2(string))))
         if(.not.istatus)call lib$stop(%val(istatus))
C.....................................................................
C     Read garbage from the directory file
C
         open(unit=17,status='old',file='transfer.tmp',form='formatted',
     1       carriagecontrol='list')
         read(17,*)                        ! skip empty line
         read(17,*)                        ! skip dir name line
         read(17,*)                        ! skip empty line
      end if
      read(17,10010,err=10)(string(j:j),j=1,21)
10010 format(21a1)
      if(string(1:1).eq.' ')go to 10
      dev_flag = index(ifile,':')
      do j = 1 , dev_flag
         wfile(j:j) = ifile(j:j)
      end do
      do j = 1,(index(string,';')-1)
         k = j + dev_flag
         wfile(k:k) = string(j:j)
      end do
      return
10    stop 'No more files match the input spec.'
      end
```

B9 - Routine for Data Input from File

```
C*****************************************************************
C                     Subroutine NAME: READER                    *
C*****************************************************************
C
C                        Written by:
C                     Joseph B. Farrell
C                          DREA
C                        25 Aug. 1986
C
C                 Latest revision:  30 Aug. 1986

      subroutine reader(blocks_to_read,bytes_per_block,disk,raw_bytes,
     &  quit_flag,jrec,iblk,nblocks,quit_flag2,input_channel,
     &  disk_input_channel,blocks_per_set)

      IMPLICIT INTEGER*4 (a-z)

      byte              raw_byte_data(50000)

      integer*4         blocks_to_read
      integer*4         blocks_to_skip
      integer*2         bytes_per_block
      integer*4         disk
      integer*2         input_channel
      integer*4         disk_input_channel
      integer*4         raw_bytes
      integer*2         text_iosb(4)
      integer*4         quit_flag
```

```
      integer*4         quit_flag2
      integer*4         jrec
      integer*4         iblk
      integer*4         nblocks

      common /raw/raw_byte_data

      external   io$_skipfile,io$_readvblk,io$_skiprecord,io$_rewind

C.....................................................................
C-- Read in a frame of data from tape or disk.
C
      if(jrec.eq.1) parity_count = 0
      if(disk.ne.1)then
C.....................................................................
C-- Read from Magtape.
C
         do i = 1,blocks_to_read
            parity_flag = 0
            mov = (i-1)*bytes_per_block + 1
92          status=sys$qiow(,%val(input_channel),io$_readvblk,
     1      text_iosb,,
     1      ,%ref(raw_byte_data(Mov)),%val(bytes_per_block),,,,)
C.....................................................................
C-- Check to see if there was an error on the tape read.
C
               if(.not.status.or.text_iosb(2).eq.0)then
                    frames_stored = jrec -1
                   call set_cursor(21,1)
                    if(text_iosb(4).ne.2.and.text_iosb(4)
     &              .ne.10)then
                      parity_count = parity_count + 1
C.....................................................................
C-- If parity error on the first record, skip into the file until we get
C-- to the next block which begins with the first channel, then begin.
C
                  if(jrec.eq.1.and.i.le.blocks_per_set)then
                     type*,'  Error reading first record!!'
                     blocks_to_skip = blocks_per_set - i
                     if(blocks_to_skip.ne.0)then
                        status=sys$qiow(,%val(input_channel),
     &                    io$_skiprecord,text_iosb,,,
     &                    %val(blocks_to_skip),,,,,)
                       if(.not.status)call lib$stop(%val(status))
                     end if
                      go to 92
                   end if
                     type*,'  Error reading record!!  Error count=',
     &                parity_count
C.....................................................................
C-- If a parity error occurs on a record , go back and reread the last
C-- good block of data which begins with the correct channel.
C
                      parity_flag = parity_flag + 1
                     blocks_to_skip = -(blocks_per_set + i)
                     status=sys$qiow(,%val(input_channel),
     &                io$_skiprecord,text_iosb,,,
     &                %val(blocks_to_skip),,,,,)
                     if(.not.status)call lib$stop(%val(status))
                      go to 92
               else
                 type*,'End of file encountered.'
                  if (raw_bytes.ne.0)then
                     quit_flag = 1
                     return
                  end if
               end if
              type*,'Saving ',frames_stored,' frames and exiting.'
               quit_flag2 = 1
              return
            end if
             raw_bytes = raw_bytes + bytes_per_block
                       if(parity_flag.ne.0)then
                  blocks_to_skip = parity_flag * blocks_per_set
                 status=sys$qiow(,%val(input_channel),
     &            io$_skiprecord,text_iosb,,,
     &            %val(blocks_to_skip),,,,,)
                 if(.not.status)call lib$stop(%val(status))
               end if
         end do
      else
C.....................................................................
C--   Check to see if the next read will put us past the end of the file.
C--   The -1 is to account for the fact that DBREAD counts from 0 rather
C--   than 1.)  (Remember that we haven't read block iblk yet.)
C
         if((iblk+blocks_to_read).gt.(nblocks-1))then
C.....................................................................
C--   If we're out of data, dump what we've collected, or simply close
C--   the files if we have no data to dump.
C
      write(5,*) 'Read puts us past EOF. blks to read.',blocks_to_read
```

```
      write(5,*)'iblk & nblks',iblk,nblocks
             blocks_to_read = nblocks - iblk
      write(5,*)'New blks to read',Blocks_to_read
      pause
              type*,'  End of file encountered.'
              if(blocks_to_read.gt.0)then
                   raw_bytes = blocks_to_read * bytes_per_block
                   quit_flag = 1
                  go to 33
              else
                   if(j.ne.1) quit_flag2 = 2
                   return
              end if
          else
c.......................................................................
c-- Read from Disk file.
c
              raw_bytes = blocks_to_read * bytes_per_block
33            call dbread(disk_input_channel,raw_byte_data,blocks_to_read,
     &        ierror,iblk)
              if(ierror.ne.0)then
                  type*,'Error encountered on disk read.'
                  type*,'Saving what I can and exiting.'
                  quit_flag2 = 1
                   return
              end if
              iblk=iblk+blocks_to_read
              call dbwait(disk_input_channel)
          end if
      end if
      return
      end
```

B10 - Transfer Status Routine

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c   Subroutine : TRANSFER_STATUS
c
c   Created : Summer, 1982 by U.Vic Physics Co-op student Laurie Bunch
c
c   Major Modifications : Spring, 1986 by Joe Farrell - Deep Water Acoustics
c
c   Purpose : To display to the user the major parameters in the
c             Surveillance Acoustics TRANSFER program during execution.
c
c   Called by : TRANSFER
c
c   Calls : 1) DATE_TIME - Subroutine which obtains the system date
c                          and time in an ASCII format.
c           3) ERASE_SCREEN - Subroutine which erases the screen from
c                             the specified position to the end.
c           4) SET_CURSOR - Subroutine which sets the cursor to a
c                           specified position on the terminal screen.
c           5) FORSSECNDS - Fortran library routine which determines
c                           the number of seconds difference between
c                           the number specified and the current time.
c           6) COMPRESS    - Subroutine which compresses the chosen
c                           channels into a format suitable for display.
c......................................................................
c---   Parameter and variable definitions.
c
c   REQUIRED PARAMETERS:
c        CHANNELS_CHOSEN - Channels corresponding to the hydrophones
c                          chosen for study.
c        nrec - Number of input file records processed.
c        FILES - Input and output files.
c        HYDROPHONES_CHOSEN - Hydrophones chosen for study.
c        NUMBER_CHOSEN - Number of hydrophones chosen for study.
c        number_of_frames - Upper limit placed by the user on the
c                           number of frames to be processed.
c        frames_procesed - Number of frames which have been processed
c        ALABEL         - ASCII label from input tape.
c        TIME_OFFSET - Time in seconds which the user wishes to step
c                      into the input data.
c        TOTAL_CHANNELS - Total number of channels on the input file.
c
c   INTERNAL VARIABLES:
c        AVERAGE_TIME_per frame - Amount of system time (not CPU) that
c                                 one frame requires.
c        BASE - The number of rows required to print all major
c               parameters up to the channels chosen.
c        BASE_TIME - Time zero when the timer was initiated.
c        DATE_AND_TIME - System date and time in an ASCII format.
c        DELTA_TIME - Time from base time to present.
c        HMS_TIME_OFFSET - Hours, minutes and seconds equivalent of
c                          the time offset specified by the user in
c                          seconds.
c......................................................................
```

```
C---   Main code
C
        SUBROUTINE TRANSFER_STATUS(ALABEL,TIN,TOUT,
     &                             NUMBER_CHOSEN,
     &                             TOTAL_CHANNELS,CHANNELS_CHOSEN,
     &                             HYDROPHONES_CHOSEN,Number_of_frames)
C.....................................................................
C---   Parameter and internal variable declarations.
C
        CHARACTER*320     ALABEL
        CHARACTER         DATE_AND_TIME*20
        CHARACTER*64      FILES(2)
        CHARACTER*128     OUTVECT
        CHARACTER*8       TIN
        CHARACTER*8       TOUT

        INTEGER*4         NUMBER_CHOSEN     !Must be before adjustable array.

        INTEGER*2         CHANNELS_CHOSEN(NUMBER_CHOSEN)
        INTEGER*2         HYDROPHONES_CHOSEN(NUMBER_CHOSEN)
        INTEGER*2         NELS
        INTEGER*2         total_channels

        INTEGER*4         BASE
        INTEGER*4         frames_processed
        INTEGER*4         number_of_frames
        INTEGER*4         FIRST
        INTEGER*4         HMS_TIME_OFFSET(3)

        REAL*4            average_time_per_record
        REAL*4            BASE_TIME
        REAL*4            DELTA_TIME
        REAL*4            nrec
        REAL*4            PERCENT_OVERLAP
        REAL*4            TIME_OFFSET
C.....................................................................
C---   I/O files and statistics passed by common for ease.
C
        COMMON /IO_STATISTICS/frames_processed,nrec
        COMMON /FILER/FILES
        COMMON /OUT/OUTVECT
C.....................................................................
C---   Obtain the current system time.
C
        CALL DATE_TIME(DATE_AND_TIME)
C.....................................................................
C---   If first call to this subroutine print everything to the terminal.
C
        IF (frames_processed .EQ. 1) THEN
C---------------------------------------------------------------------
C---   Print the header.
C
          CALL ERASE_SCREEN(1,1)
          TYPE 1000
1000      FORMAT('+',T29,'FILE TRANSFER STATISTICS')
          TYPE 1100, DATE_AND_TIME
1100      FORMAT(' ',T31,A)
          TYPE 1200
1200      FORMAT(' ',80('-'))

C---------------------------------------------------------------------
C---   Print the ASCII label and the input and output files.
C
          TYPE 2000, ALABEL(65:110)
2000      FORMAT(T3,'ASCII label : ',A46)
          CALL SET_CURSOR(5,1)
           type 3000, files(2)(1:24)
3000       format(/t42,'Output file :  ',A)
           type 3001, files(1)(1:24)
3001       format('+',t3,'Input file :  ',A)
C---------------------------------------------------------------------
C---   Print the time offset in its new form.
C
          TYPE 4000, TIN,TOUT
4000      FORMAT (/T3,'Starts at   :  ',A,T42,'Starts at   :  ',A)
C---------------------------------------------------------------------
C---   If the number of frames is greater than 5000,then the program is
C---    running till it hits the end of the input file (EOF).
C
          IF (number_of_frames .GE. 3000) THEN
            TYPE 5000,'TO EOF'
5000        FORMAT(/T3,'Number of frames requested : ',A6)
          ELSE
            TYPE 5001,number_of_frames
5001        FORMAT(/T3,'Number of frames requested : ',I6)
          ENDIF
C---------------------------------------------------------------------
C---   Print the number of channels out of the total number which are
C---   being studied. Show also the channels and the hydrophones they
C---   correspond to.
C
          TYPE 6000, NUMBER_CHOSEN,TOTAL_CHANNELS
```

```
6000        FORMAT(/' ',T3,'Channel usage : ',I2,' chosen out of ',
     &              I2,' total')
C..  ..........................................................
C---   Call routine to compress the chosen channels into a format
C---   suitable for screen output, ie. 1-5,7,11,15-24.
C
         CALL COMPRESS(Number_chosen,Channels_chosen,NELS)
          TYPE*,'   Channels :', OUTVECT(:NELS)
         CALL COMPRESS(Number_chosen,Hydrophones_chosen,NELS)
           TYPE*,'   Acoustic Channels:', OUTVECT(:NELS)
C-------------------------------------------------------------------------
C---   Print the number of records used, the number of FFT's calculated,
C---   and the average amount of system time per FFT per channel (not
C---   known on the first call.)
C
           TYPE 7000, frames_processed
7000       FORMAT(/' ',T3,'Number of input file accesses : ',I6)
           TYPE 8000, nrec
8000       FORMAT(/' ',T3,'Number of records processed : ',f6.2)
           TYPE 8100
8100       FORMAT(' ',T3,'Average time per frame :       sec')

C-------------------------------------------------------------------------
C---   Set the timer and find the absolute value of time zero.
C
           BASE_TIME = FORSSECNDS(0.0)
C.....................................................................
C---   For successive calls just print the dynamic paramters.
C
         ELSE
C-------------------------------------------------------------------------
C---   Find the amount of system time to have elapsed since time zero.
C
           DELTA_TIME = FORSSECNDS(BASE_TIME)
C-------------------------------------------------------------------------
C---   Calculate the average time per FFT per channel.
C
           average_time_per_record = DELTA_TIME / nrec
C-------------------------------------------------------------------------
C---   Update the system time.
C
           CALL SET_CURSOR(2,30)
           TYPE 9000, DATE_AND_TIME
9000       FORMAT('+',A)
C-------------------------------------------------------------------------
C---   If the number of channels and hydrophones printed on the first call
C---   were less than or equal to 16 then 16 lines were devoted to static
C---   parameters; otherwise, 18 lines were devoted to static parameters.
C
           IF (NELS .LE. 50) THEN
             BASE = 15
           ELSE
             BASE = 17
           ENDIF
C-------------------------------------------------------------------------
C---   Update the number of records used, the FFT's completed and the
C---   average time per FFT per channel.
C
           CALL SET_CURSOR(BASE + 2,37)
           TYPE 9100, frames_processed
9100       FORMAT('+',I6)
           CALL SET_CURSOR(BASE + 4,33)
           TYPE 9200, nrec
9200       FORMAT('+',f6.1)
           CALL SET_CURSOR(BASE + 5,27)
           TYPE 9300, average_time_per_record
9300       FORMAT('+',F5.2)
         ENDIF

         RETURN
         END

         SUBROUTINE COMPRESS(NUMBER_CHOSEN,INVECT,NELS)

         CHARACTER*128 OUTVECT

         INTEGER*2          Number_chosen
         INTEGER*2          INVECT(NUMBER_CHOSEN)
         INTEGER 2          Last_flag
         INTEGER*2          START
         INTEGER*2          ST
         INTEGER*2          Temp_flag
         INTEGER*2          NELS
         INTEGER*2          L
         INTEGER*2          M

         COMMON /OUT/OUTVECT

         if(number_chosen.eq.1)then
            encode(2,100,outvect(1:2))invect(1)
100         format(i2)
            nels = 2
```

```
      return
    end if
    START = INVECT(1)
    ST = START
    NELS = 0
    Last_flag = 0
    Temp_flag = 0
    L = 1
    M = 2
    DO J = 2,Number_chosen
       IF(J.EQ.Number_chosen)Last_flag = 1
       IF(Last_flag.EQ.1)THEN
          IF(INVECT(J).EQ.(ST+1))THEN
             ST = ST + 1
             CALL BUILDER(L,M,NELS,Last_flag,Start,St
          ELSE
             CALL BUILDER(L,M,NELS,Temp_flag,Start,St
             ST = INVECT(J)
             CALL BUILDER(L,M,NELS,Last_flag,ST,ST
          END IF
       ELSE
          IF(INVECT(J).EQ.(ST+1))THEN
             ST = ST + 1
          ELSE
             CALL BUILDER(L,M,NELS,Last_flag,Start,St
             START = INVECT(J)
             ST = START
          END IF
       END IF
    END DO
    RETURN
    END

    SUBROUTINE BUILDER(L,M,NELS,Last_flag,Start,STOP

    CHARACTER*128      OUTVECT
    CHARACTER*1        SEPARATOR

    INTEGER*2          Last_flag
    INTEGER*2          START
    INTEGER*2          STOP
    INTEGER*2          More_flag
    INTEGER*2          NELS
    INTEGER*2          L
    INTEGER*2          M

    COMMON /OUT/OUTVECT

    IF(START.EQ.STOP)THEN
       ENCODE(2,100,OUTVECT(L:M))START
100    FORMAT(I2)
       SEPARATOR = ','
       CALL FILLER(START,L,M,SEPARATOR,LAST_FLAG,NELS
    ELSE
       ENCODE(2,100,OUTVECT(L:M))START
       SEPARATOR = '-'
       CALL FILLER(START,L,M,SEPARATOR,LAST_FLAG,NELS
       ENCODE(2,100,OUTVECT(L:M))STOP
       SEPARATOR = ','
       CALL FILLER(STOP,L,M,SEPARATOR,LAST_FLAG,NELS
    END IF
    RETURN
    END

    SUBROUTINE FILLER(ELEMENT,L,M,SEPARATOR,LAST_FLAG,NELS

    CHARACTER*128      OUTVECT
    CHARACTER*1        SEPARATOR

    INTEGER*2          Last_flag
    INTEGER*2          ELEMENT
    INTEGER*2          NELS
    INTEGER*2          L
    INTEGER*2          M

    COMMON /OUT/OUTVECT

    IF(ELEMENT.LT.10)THEN
       OUTVECT(L:L) = OUTVECT(M:M)
       IF(Last_flag.EQ.1.AND.SEPARATOR.EQ.     THEN
          NELS = NELS + 1
       ELSE
          OUTVECT(M:M) = SEPARATOR
          NELS = NELS + 2
          L = L + 2
          M = M + 2
       END IF
     ELSE
       IF(Last_flag.EQ.1.AND.SEPARATOR.EQ.     THEN
          NELS = NELS + 2
       ELSE
          OUTVECT((M+1):(M+1)) = SEPARATOR
```

```
        NELS = NELS + 3
        I = I + 3
        M = M + 3
    END IF
END IF
RETURN
END
```

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATING ACTIVITY: DREA	2a. DOCUMENT SECURITY CLASSIFICATION: UNCLASSIFIED 2b. GROUP: TC

3. DOCUMENT TITLE: A VERSATILE TOOL FOR DATA FILE TRANSFER AND MANIPULATION

4. DESCRIPTIVE NOTES (Type of report and inclusive dates): Technical Communication, Dec. 1986

5. AUTHOR(S) (Last name, first name, middle initial): FARRELL, Joseph B.

6. DOCUMENT DATE: January 1987	7a. TOTAL NO. OF PAGES: 57	7b. NO. OF REFS: 3
8a. PROJECT OR GRANT NO.	9a. ORIGINATOR'S DOCUMENT NUMBER(S): DREA TECHNICAL COMMUNICATION 87/303	
8b. CONTRACT NO.	9b. OTHER DOCUMENT NO.(S) (Any other numbers that may be assigned this document)	

10. DISTRIBUTION STATEMENT: UNLIMITED DISTRIBUTION

11. SUPPLEMENTARY NOTES	12. SPONSORING ACTIVITY

13. ABSTRACT

This document describes in detail a software tool for manipulating data files. The Surveillance Acoustics section at Defence Research Establishment Atlantic has acquired VAX computers over the last few years, and analysis tasks which were formerly done on PDP-11 computers are now being moved to the VAXen. PDP-11s are still used in the at-sea data collection role, so some means is necessary of transferring the data files thus produced to the VAXen for signal processing and analysis. PDP-11 data files are typically located on 9-track magnetic tape, so one method of transferring the data would be to read PDP-11 tapes on the VAXen. The software tool described here (a program named **TRANSFER**) was written, in part, to perform this data transfer chore, taking into account the special formats and header information in the files produced by the PDP-11s. Manipulation of data files already residing on a VAX is also possible using TRANSFER. The program is versatile, allowing the user to choose channels and data segments to be transferred between files with a high degree of freedom.

KEY WORDS

Computers

VAX

File Transfer

Data Manipulation

Underwater Acoustics

INSTRUCTIONS

1 ORIGINATING ACTIVITY Enter the name and address of the organization issuing the document.

2a DOCUMENT SECURITY CLASSIFICATION Enter the overall security classification of the document including special warning terms whenever applicable.

2b. GROUP Enter security reclassification group number. The three groups are defined in Appendix 'M' of the DRB Security Regulations.

3 DOCUMENT TITLE Enter the complete document title in all capital letters. Titles in all cases should be unclassified. If a sufficiently descriptive title cannot be selected without classification, show title classification with the usual one-capital-letter abbreviation in parentheses immediately following the title.

4 DESCRIPTIVE NOTES Enter the category of document, e.g. technical report, technical note or technical letter. If appropriate, enter the type of document, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.

5 AUTHOR(S): Enter the name(s) of author(s) as shown on or in the document. Enter last name, first name, middle initial. If military, show rank. The name of the principal author is an absolute minimum requirement.

6. DOCUMENT DATE: Enter the date (month, year) of Establishment approval for publication of the document.

7a TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b NUMBER OF REFERENCES Enter the total number of references cited in the document.

8a PROJECT OR GRANT NUMBER If appropriate, enter the applicable research and development project or grant number under which the document was written.

8b CONTRACT NUMBER If appropriate, enter the applicable number under which the document was written.

9a ORIGINATOR'S DOCUMENT NUMBER(S) Enter the official document number by which the document will be identified and controlled by the originating activity. This number must be unique to this document.

9b. OTHER DOCUMENT NUMBER(S). If the document has been assigned any other document numbers (either by the originator or by the sponsor), also enter this number(s).

10. DISTRIBUTION STATEMENT Enter any limitations on further dissemination of the document, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this document from their defence documentation center."

(2) "Announcement and dissemination of this document is not authorized without prior approval from originating activity."

11. SUPPLEMENTARY NOTES Use for additional explanatory notes.

12. SPONSORING ACTIVITY Enter the name of the departmental project office or laboratory sponsoring the research and development. Include address.

13. ABSTRACT Enter an abstract giving a brief and factual summary of the document, even though it may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall end with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (TS), (S), (C), (R), or (U).

The length of the abstract should be limited to 20 single-spaced standard typewritten lines; 7½ inches long.

14 KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a document and could be helpful in cataloging the document. Key words should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context.

END

5-87

DTIC